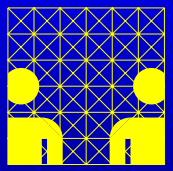


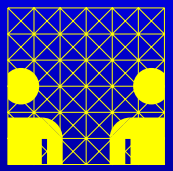
# Erweiterte Anfragesprachen für RACER

## Work in Progress

### *STS-Vortrag*

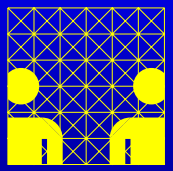
Michael Wessel

Arbeitsbereich Kognitive Systeme
Universität Hamburg
Projekt “DLS”



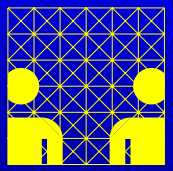
# Inhalt

- Erinnerung: Anfragen in RACER



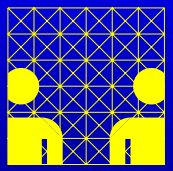
# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.



# Inhalt

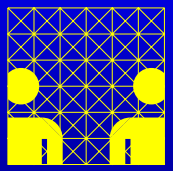
- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

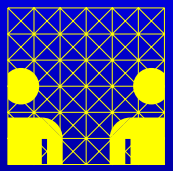
  $\Rightarrow$ Erweiterungswünsche



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

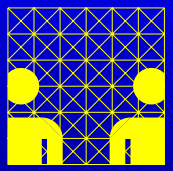
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

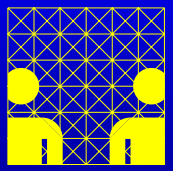
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

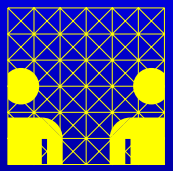
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen
  - Erweiterung: Disjunktion und Negation



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

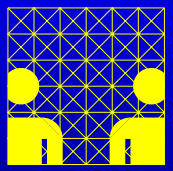
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen
  - Erweiterung: Disjunktion und Negation
- Implementation



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

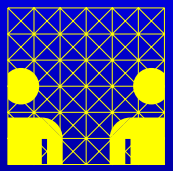
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen
  - Erweiterung: Disjunktion und Negation
- Implementation
  - Parsing, Repräsentation, Optimierung



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

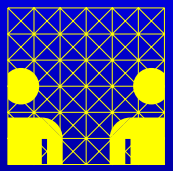
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen
  - Erweiterung: Disjunktion und Negation
- Implementation
  - Parsing, Repräsentation, Optimierung
  - Übersetzung, Ausführung



# Inhalt

- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

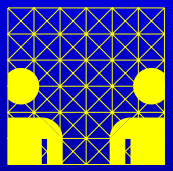
  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen
  - Erweiterung: Disjunktion und Negation
- Implementation
  - Parsing, Repräsentation, Optimierung
  - Übersetzung, Ausführung
  - Inferenz mit Anfragen



# Inhalt

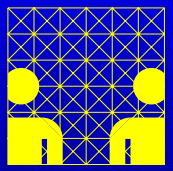
- Erinnerung: Anfragen in RACER
  - Intensionale Anfragen vs.
  - Extensionale Anfragen

  $\Rightarrow$ Erweiterungswünsche
- Erweiterte Anfragemöglichkeiten f. RACER
  - Konjunktive Anfragen
  - Erweiterung: Disjunktion und Negation
- Implementation
  - Parsing, Repräsentation, Optimierung
  - Übersetzung, Ausführung
  - Inferenz mit Anfragen
  - “Query Repository”



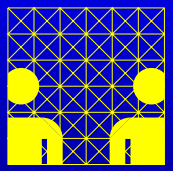
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)



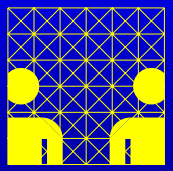
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
  - `(and mother`
    `(some has-child`
    `(some has-child person)))`



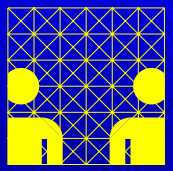
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
  - `(and mother`<br>`(some has-child`<br>`(some has-child person)))`
  - $mother \sqcap \exists has\_child.\exists has\_child.person$



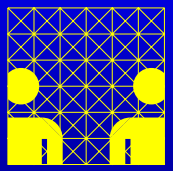
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie



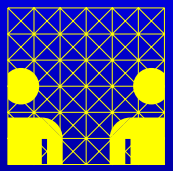
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $\quad (\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$



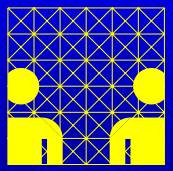
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$



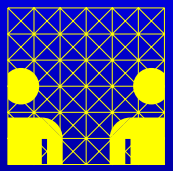
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - `(define-primitive-concept man human)`



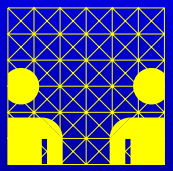
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - `(define-primitive-concept man human)`
  - `(define-concept man (and human (some has-gender male)))`



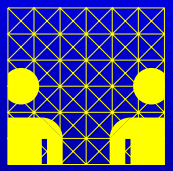
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - `(define-primitive-concept man human)`
  - `(define-concept man (and human (some has-gender male)))`
  - `(implies man human)`



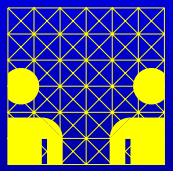
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - `(define-primitive-concept man human)`
  - `(define-concept man (and human (some has-gender male)))`
  - `(implies man human)`
  - `(equivalent man (and human (some has-gender male)))`



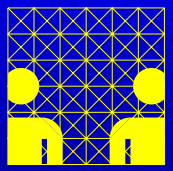
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - Axiome f. Rollen-Beziehungen



# Wiederholung: DL-Begriffe

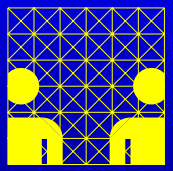
- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - Axiome f. Rollen-Beziehungen
  - 
```
(define-primitive-role has-child
        :parent has-descendant
        :inverse has-parent
        :domain parent
        :range person)
```



# Wiederholung: DL-Begriffe

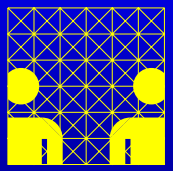
- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
  - $\boxed{C \sqsubseteq D}$ $(\forall x : C(x) \Rightarrow D(x), C^{\mathcal{I}} \subseteq D^{\mathcal{I}})$
  - $\boxed{C \equiv D}$ $(\forall x : C(x) \Leftrightarrow D(x), C^{\mathcal{I}} = D^{\mathcal{I}})$
  - Axiome f. Rollen-Beziehungen
  - $\forall x, y :$

$$
\begin{array}{l}
( \ (has\_child(x,y) \Leftrightarrow has\_parent(y,x)) \wedge \\
\quad (has\_child(x,y) \Rightarrow \\
\qquad\qquad ( \ has\_descendant(x,y) \wedge \\
\qquad\qquad\quad parent(x) \wedge person(y))))
\end{array}
$$



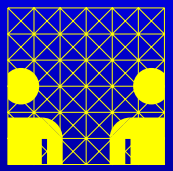
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$



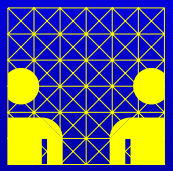
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$



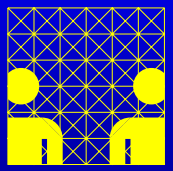
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$
  - `(instance betty mother)`



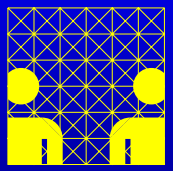
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer "konkreten Welt"
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$
  - `(instance betty mother)`
  - `(related betty eve has-child)`



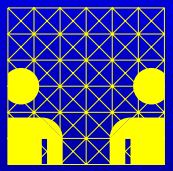
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$
  - ABoxen haben i.d.R. mehr als ein Modell (z.B. $\{i : C \sqcup D\}$)



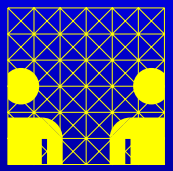
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$
  - ABoxen haben i.d.R. mehr als ein Modell (z.B. $\{i : C \sqcup D\}$)
  - Open Domain Assumption (z.B. $\{i : \exists R.C\}$)



# Wiederholung: DL-Begriffe

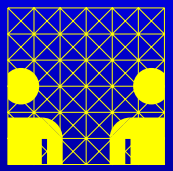
- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$
  - ABoxen haben i.d.R. mehr als ein Modell (z.B. $\{i : C \sqcup D\}$)
  - Open Domain Assumption (z.B. $\{i : \exists R.C\}$)
  - Open World Assumption (kein “Negation as Failure”)



# Wiederholung: DL-Begriffe

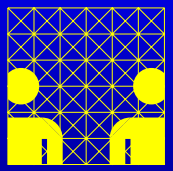
- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer "konkreten Welt"
  - $\boxed{i : C}$ $\quad (C(i), i^{\mathcal{I}} \in C^{\mathcal{I}})$
  - $\boxed{(i, j) : R}$ $\quad (R(i, j), (i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}})$
  - ABoxen haben i.d.R. mehr als ein Modell (z.B. $\{i : C \sqcup D\}$)
  - Open Domain Assumption (z.B. $\{i : \exists R.C\}$)
  - Open World Assumption (kein "Negation as Failure")

  $\Rightarrow$ ABoxen sind keine Datenbanken



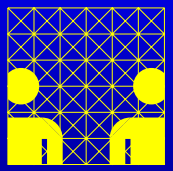
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
- Wissensbasis = (TBox,ABox)



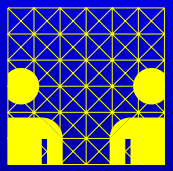
# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
- Wissensbasis = (TBox,ABox)
- Taxonomie = klassifizierte TBox



# Wiederholung: DL-Begriffe

- Konzept(term): logische Formel, Wort einer DL, z.B. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (RACER-DL)
- TBox: Modellierung der Ontologie
- ABox: Modellierung einer “konkreten Welt”
- Wissensbasis = (TBox,ABox)
- Taxonomie = klassifizierte TBox
- Ontologie $\approx$ TBox, mit f.d. Anwendung relevanten Konzepten



# Family TBox

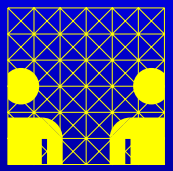
```
(implies person (and human (some has-gender (or female male))))
(disjoint female male)
(implies woman (and person (some has-gender female)))
(implies man (and person (some has-gender male)))

(equivalent parent (and person (some has-child person)))
(equivalent mother (and woman parent))
(equivalent father (and man parent))

(equivalent grandmother
            (and mother
                 (some has-child
                       (some has-child person))))

(equivalent aunt (and woman (some has-sibling parent)))
(equivalent uncle (and man (some has-sibling parent)))

(equivalent brother (and man (some has-sibling person)))
(equivalent sister (and woman (some has-sibling person)))
```



# Family TBox - Rollen

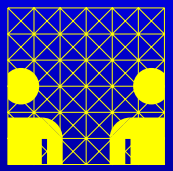
```
(define-primitive-role has-descendant
                       :transitive t
                       :inverse descendant-of)

(define-primitive-role has-child
                       :parent has-descendant
                       :inverse has-parent
                       :domain parent :range person)

(define-primitive-role has-sibling
                       :inverse has-sibling
                       :domain (or sister brother)
                       :range (or sister brother))

(define-primitive-role has-sister
                       :parent has-sibling
                       :range sister :inverse sister-of)

(define-primitive-attribute has-gender)
```



# Family ABox - “Smith Family”

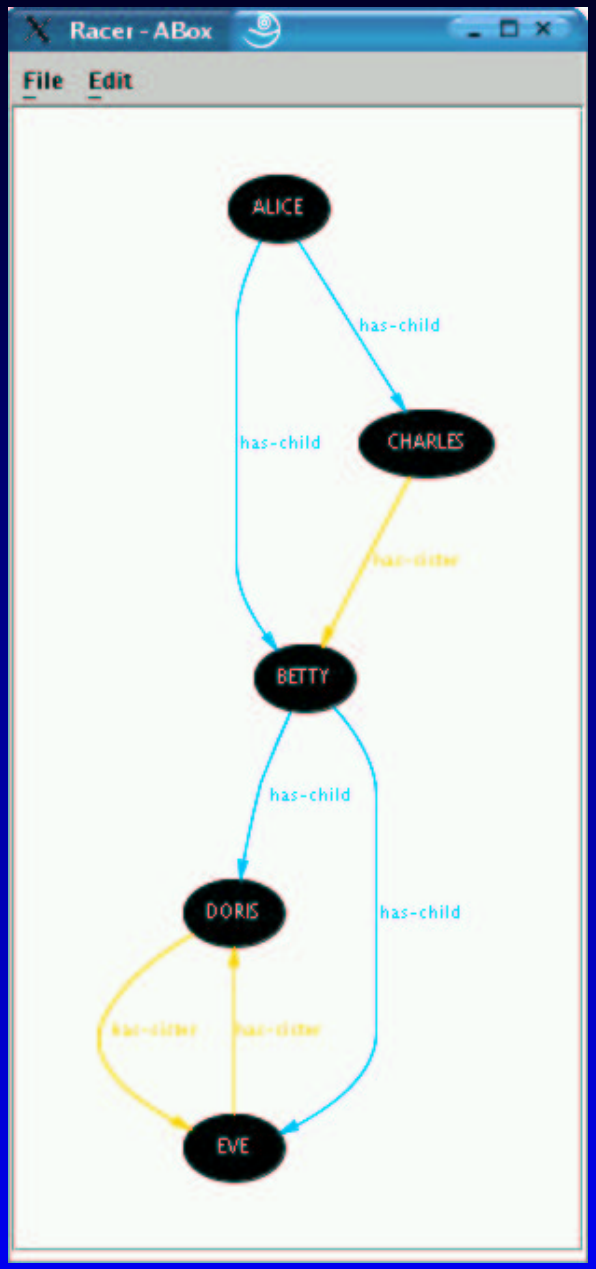
```
(instance alice mother)
(instance alice woman)
(related alice betty has-child)
(related alice charles has-child)

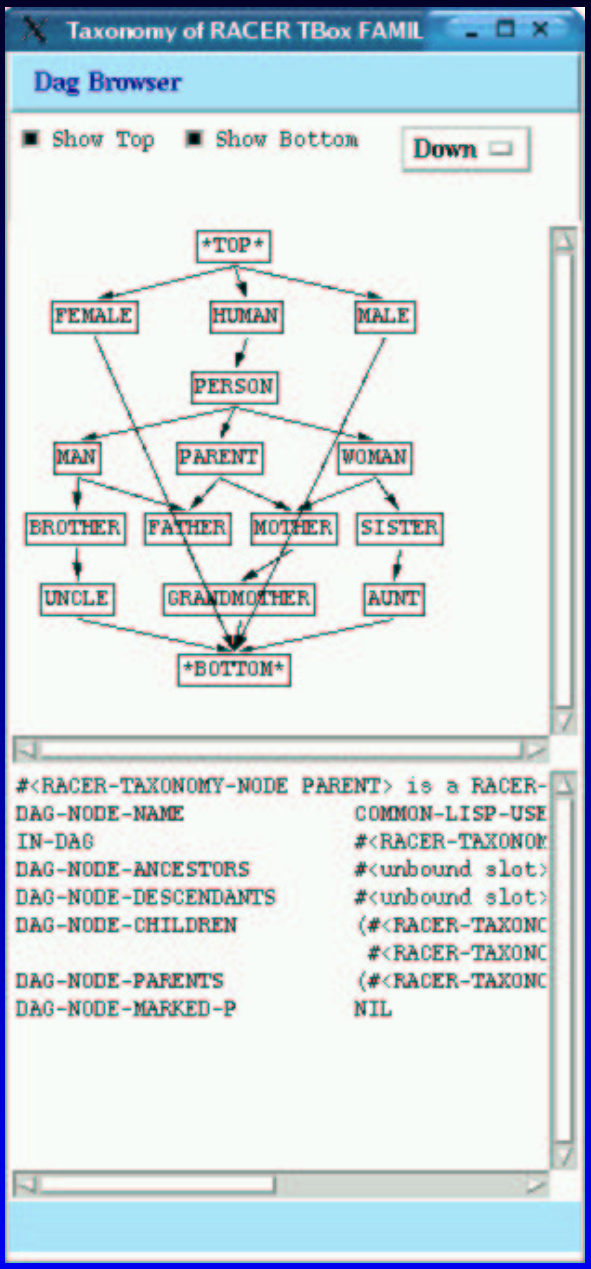
(instance betty mother)
(related betty doris has-child)
(related betty eve has-child)

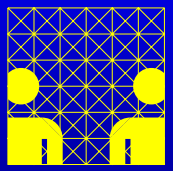
(instance charles brother)
(related charles betty has-sister)

(related doris eve has-sister)
```

# ABox-Graph, Taxonomie-DAG

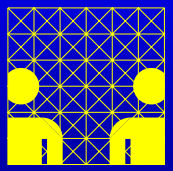




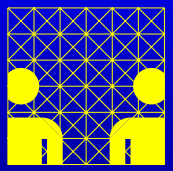
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes



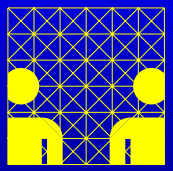
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen



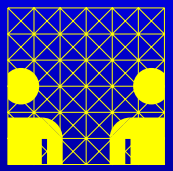
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox



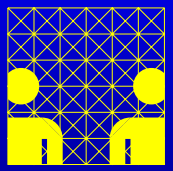
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)



# Anfragen in RACER

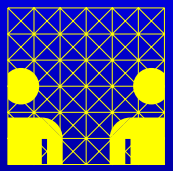
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - `(concept-satisfiable? (and woman man))`



# Anfragen in RACER

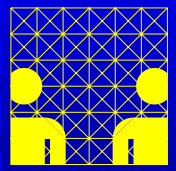
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - `(concept-satisfiable? (and woman man))`

  $\Rightarrow$ `NIL`



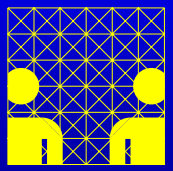
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)



# Anfragen in RACER

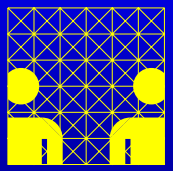
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - `(concept-subsumes? human man)`



# Anfragen in RACER

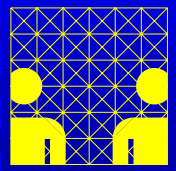
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - `(concept-subsumes? human man)`

  $\Rightarrow$ T



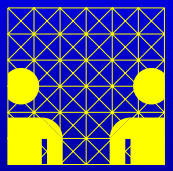
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie



# Anfragen in RACER

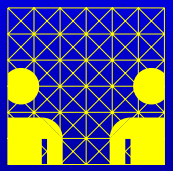
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
  - `(concept-parents uncle)`



# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
  - `(concept-parents uncle)`

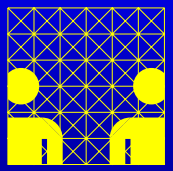
  $\Rightarrow$ `((brother))`



# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
  - `(concept-parents uncle)`

  $\Rightarrow$ `((brother))`
  - `(concept-ancestors uncle)`

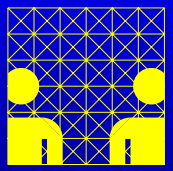


# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
  - `(concept-parents uncle)`

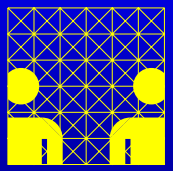
  ⇒ `((brother))`
  - `(concept-ancestors uncle)`

  ⇒ `((*top* top) (human) (person) (man) (brother))`



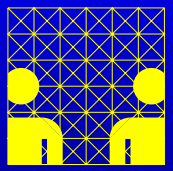
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
- Extensionale Anfragen



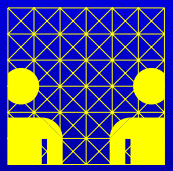
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
- Extensionale Anfragen
  - Grob: betreffen die ABox



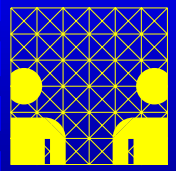
# Anfragen in RACER

- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
- Extensionale Anfragen
  - Grob: betreffen die ABox
  - Aber Bezug auf definierte Konzepte i.d. TBox



# Anfragen in RACER

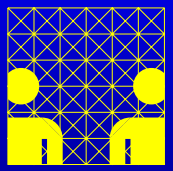
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
- Extensionale Anfragen
  - Grob: betreffen die ABox
  - Aber Bezug auf definierte Konzepte i.d. TBox
  - `(concept-instances woman)`



# Anfragen in RACER

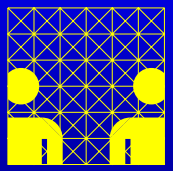
- Standard-Inferenzdienste eines DL-Systemes
- Rein intensionale Anfragen
  - Grob: betreffen Konzepte und/oder TBox
  - Zentral: Konzepterfüllbarkeit (bzgl. TBox)
  - Konzeptsubsumption (bzgl. TBox)
  - Einige Anfragen erfordern Taxonomie
- Extensionale Anfragen
  - Grob: betreffen die ABox
  - Aber Bezug auf definierte Konzepte i.d. TBox
  - `(concept-instances woman)`

  $\Rightarrow$ `(alice betty doris eve)`



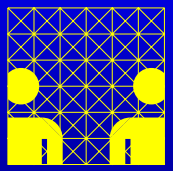
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid\ (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$



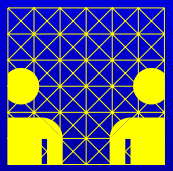
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg \mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$



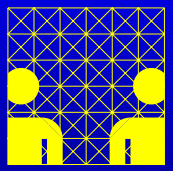
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid\ (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg \mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption



# Eigenschaften von `concept-instances`

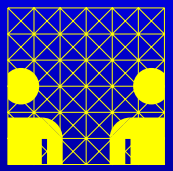
- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
  - `(concept-instances grandFATHER)`



# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
  - `(concept-instances grandFATHER)`

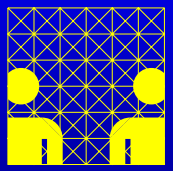
  $\Rightarrow$ `NIL`



# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
  - `(concept-instances grandFATHER)`

  $\Rightarrow$ `NIL`
  - `(concept-instances (not grandFATHER))`

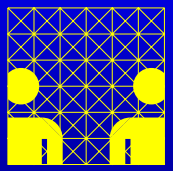


# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
  - `(concept-instances grandFATHER)`

  $\Rightarrow$ `NIL`
  - `(concept-instances (not grandFATHER))`

  $\Rightarrow$ `DL-System: NIL`



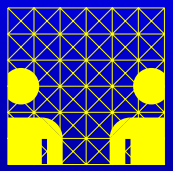
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
  - `(concept-instances grandFATHER)`

  $\Rightarrow$ `NIL`
  - `(concept-instances (not grandFATHER))`

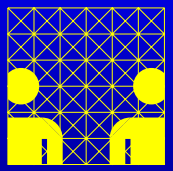
  $\Rightarrow$ `DL-System: NIL`

  $\Rightarrow$ `CWA oder NAF: (charles alice betty eve doris)`



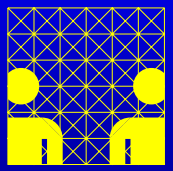
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid\ (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption



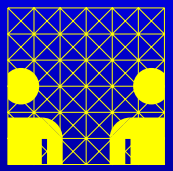
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg \mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
  - $\mathfrak{A} = \{i : \exists R.C\}$



# Eigenschaften von `concept-instances`

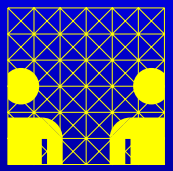
- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
  - $\mathfrak{A} = \{i : \exists R.C\}$
  - `(concept-instances C)`



# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
  - $\mathfrak{A} = \{i : \exists R.C\}$
  - `(concept-instances C)`

  $\Rightarrow$ `DL-System: NIL`

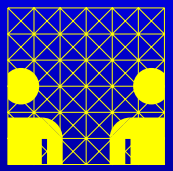


# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
  - $\mathfrak{A} = \{i : \exists R.C\}$
  - `(concept-instances C)`

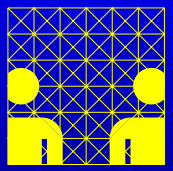
  $\Rightarrow$ `DL-System: NIL`

  $\Rightarrow$ `alternativ: (#:temp-123)`



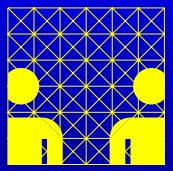
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg \mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig



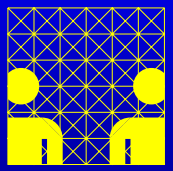
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
  - Anfrage “(Onkel,Geschwister)-Paare”?



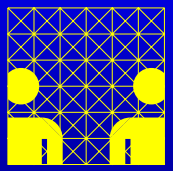
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
  - Anfrage “(Onkel,Geschwister)-Paare”?
  - `(concept-instances uncle)` $\Rightarrow$ `(charles)`



# Eigenschaften von `concept-instances`

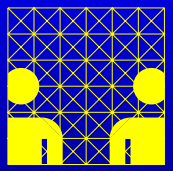
- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
  - Anfrage “(Onkel,Geschwister)-Paare”?
  - `(concept-instances uncle)` $\Rightarrow$ `(charles)`
  - `(individual-fillers charles has-sibling)` $\Rightarrow$ `(betty)`



# Eigenschaften von `concept-instances`

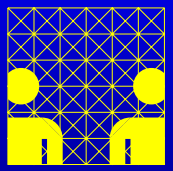
- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
  - Anfrage "(Onkel,Geschwister)-Paare"?
  - `(concept-instances uncle)` $\Rightarrow$ `(charles)`
  - `(individual-fillers charles has-sibling)` $\Rightarrow$ `(betty)`

  $\Rightarrow \{(\texttt{charles}, \texttt{betty})\}$



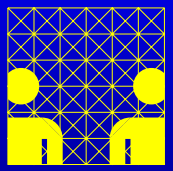
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
- Nachteil: bel. Koreferenzen nicht ausdrückbar



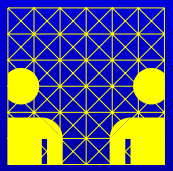
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
- Nachteil: bel. Koreferenzen nicht ausdrückbar
  - "Welche Menschen $(X, Y)$ haben die gleiche Mutter?"



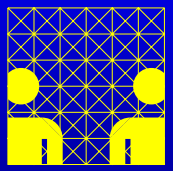
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
- Nachteil: bel. Koreferenzen nicht ausdrückbar
  - "Welche Menschen $(X, Y)$ haben die gleiche Mutter?"
  - u.d.A., dass `has-sibling`, `has-brother`, `has-sister` fehlen in der WB



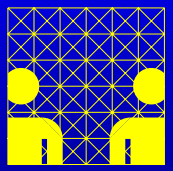
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
- Nachteil: bel. Koreferenzen nicht ausdrückbar
- Schön wären “Conjunctive Queries”
  - `answer(x,y)` $\leftarrow$ `uncle(x)` $\wedge$ `has-sibling(x,y).`



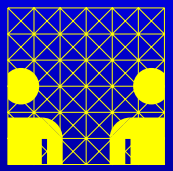
# Eigenschaften von `concept-instances`

- $concept\_instances(C) =_{def} \{\, i \in \mathfrak{A} \mid (\mathfrak{T}, \mathfrak{A}) \models (i : C) \,\}$
- $(\mathfrak{T}, \mathfrak{A}) \models i : C$ iff $\neg\mathsf{SAT}((\mathfrak{T}, \mathfrak{A} \cup \{i : \neg C\}))$
- Open World Assumption
- Active Domain Assumption
- Nachteil: Anfragen sind einstellig
- Nachteil: bel. Koreferenzen nicht ausdrückbar
- Schön wären “Conjunctive Queries”
  - `answer(x,y)` $\leftarrow$ `uncle(x)` $\wedge$ `has-sibling(x,y).`
  - `answer(x,y)` $\leftarrow$ `has-parent(x,m)` $\wedge$ `has-parent(y,m)` $\wedge$ `mother(m).`



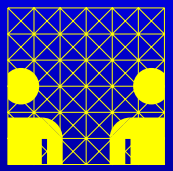
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”



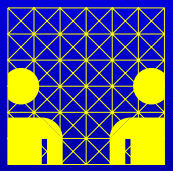
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei



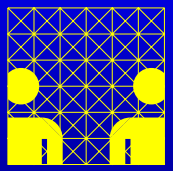
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome



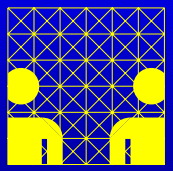
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme



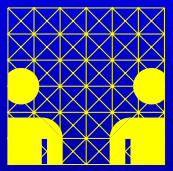
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme



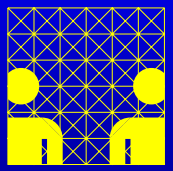
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen



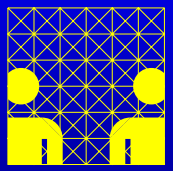
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries



# “Conjunctive Queries” für RACER

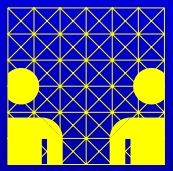
- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries
  - `answer()` ← `woman(betty).`



# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries
  - `answer()` ← `woman(betty).`

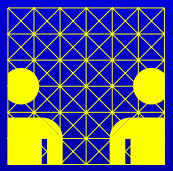
  ⇒ `T`



# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries
  - `answer()` $\leftarrow$ `woman(betty).`

  $\Rightarrow$ `T`
  - `answer(betty)` $\leftarrow$ `woman(betty).`

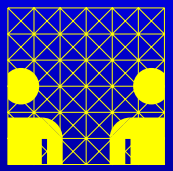


# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries
  - `answer() ← woman(betty).`

  ⇒ `T`
  - `answer(betty) ← woman(betty).`

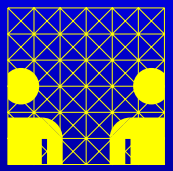
  ⇒ `((betty betty))`



# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries

⊕ konzeptionell einfach

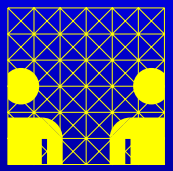


# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries

⊕ konzeptionell einfach

⊕ implementierbar als RACER “add on”



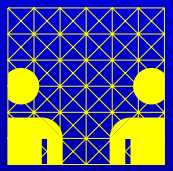
# “Conjunctive Queries” für RACER

- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries

⊕ konzeptionell einfach

⊕ implementierbar als RACER “add on”

⊕ gut optimierbar (endliches CSP)



# “Conjunctive Queries” für RACER

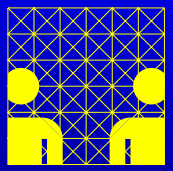
- “Straight-Forward Extension”
- Behalte die “Active Domain”-Semantik bei
- Erlaube ein- und zweistellige Atome
  - unäre Atome = Konzeptterme
  - binäre Atome = Rollenterme
- Erlaube beliebig viele Variablen
- Erlaube Nominals in den Queries

⊕ konzeptionell einfach

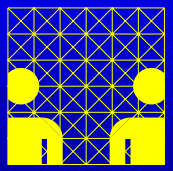
⊕ implementierbar als RACER “add on”

⊕ gut optimierbar (endliches CSP)

⊕ Eigenschaften der Atome in WB definiert

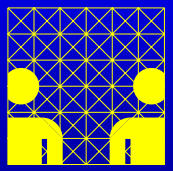


# Kurze Demo



# Semantik der Queries

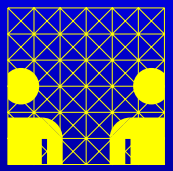
- ```
  (and (?x woman)
       (?y human)
       (?z man)
       (?x ?y has-child)
       (?y ?z has-sibling))
  ```
  ```



# Semantik der Queries

- 
```
(and (?x woman)
     (?y human)
     (?z man)
     (?x ?y has-child)
     (?y ?z has-sibling))
```

- $\exists x, y, z : \; (x \neq y) \wedge (x \neq z) \wedge (y \neq z) \wedge$
  $woman(x) \wedge human(y) \wedge man(z) \wedge$
  $has\_child(x, y) \wedge has\_sibling(y, z)$



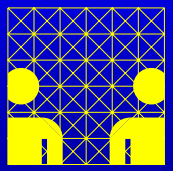
# Semantik der Queries

- ```
  (and (?x woman)
       (?y human)
       (?z man)
       (?x ?y has-child)
       (?y ?z has-sibling))
  ```
- $answer(x, y, z) =_{def}$

  $(x \neq y) \wedge (x \neq z) \wedge (y \neq z) \wedge$

  $woman(x) \wedge human(y) \wedge man(z) \wedge$

  $has\_child(x, y) \wedge has\_sibling(y, z)$
  ```

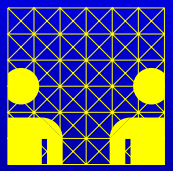


# Semantik der Queries

- ```
  (and (?x woman)
       (?y human)
       (?z man)
       (?x ?y has-child)
       (?y ?z has-sibling))
  ```
- $answer(x, y, z) =_{def}$

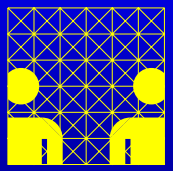
$$(x \neq y) \wedge (x \neq z) \wedge (y \neq z) \wedge$$
$$woman(x) \wedge human(y) \wedge man(z) \wedge$$
$$has\_child(x, y) \wedge has\_sibling(y, z)$$

$$\{ (i, j, k) \mid i, j, k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, (\mathfrak{A}, \mathfrak{T}) \models answer(x, y, z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k} \}$$
  ```



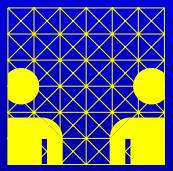
# Semantik der Queries (2)

- `has-child(betty,?x)`



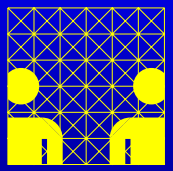
# Semantik der Queries (2)

- `has-child(betty,?x)`
- $answer(betty, x) =_{def}$
  $(x \neq betty) \wedge$
  $betty = betty\_constant \wedge$
  $has\_child(betty, x)$



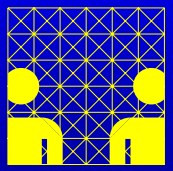
# Semantik der Queries (2)

- `has-child(betty,?x)`
- $answer(betty, x) =_{def}$
  $(x \neq betty) \wedge$
  $betty = betty\_constant \wedge$
  $has\_child(betty, x)$
- Beachte: Active Domain Semantics



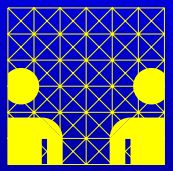
# Semantik der Queries (2)

- `has-child(betty,?x)`
- $answer(betty, x) =_{def}$
  $(x \neq betty) \wedge$
  $betty = betty\_constant \wedge$
  $has\_child(betty, x)$
- Beachte: Active Domain Semantics
- $\{i : \exists R.C\} \models \exists x, y : R(x, y) \wedge C(y)$,



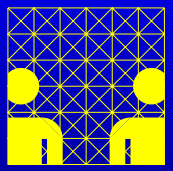
# Semantik der Queries (2)

- `has-child(betty,?x)`
- $answer(betty, x) =_{def}$
  $(x \neq betty) \wedge$
  $betty = betty\_constant \wedge$
  $has\_child(betty, x)$
- Beachte: Active Domain Semantics
- $\{i : \exists R.C\} \models \exists x, y : R(x, y) \wedge C(y)$,
- insbesondere sogar $i = j$ möglich



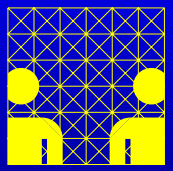
# Semantik der Queries (2)

- `has-child(betty,?x)`
- $answer(betty, x) =_{def}$
  $(x \neq betty) \wedge$
  $betty = betty\_constant \wedge$
  $has\_child(betty, x)$
- Beachte: Active Domain Semantics
- $\{i : \exists R.C\} \models \exists x, y : R(x, y) \wedge C(y)$,
- insbesondere sogar $i = j$ möglich
- `(and (?x ?y R) (?Y C))` $\Rightarrow$ `NIL`



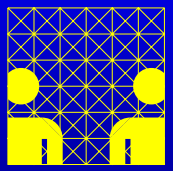
# Semantik der Queries (2)

- `has-child(betty,?x)`
- $answer(betty, x) =_{def}$
  $(x \neq betty) \wedge$
  $betty = betty\_constant \wedge$
  $has\_child(betty, x)$
- Beachte: Active Domain Semantics
- $\{i : \exists R.C\} \models \exists x, y : R(x, y) \wedge C(y)$,
- insbesondere sogar $i = j$ möglich
- `(and (?x ?y R) (?Y C))` $\Rightarrow$ `NIL`
- `(and (?x (SOME R C)))` $\Rightarrow$ `((?x i))`



# (Zu) Primitiver Algorithmus

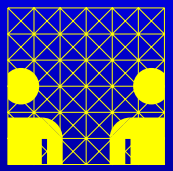
$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid\; i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k,\; (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

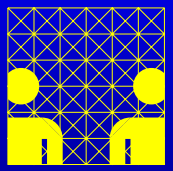
- “Generate & Test”-Algorithmus suggeriert



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

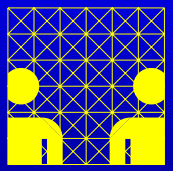
- “Generate & Test”-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid\; i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k,\; (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

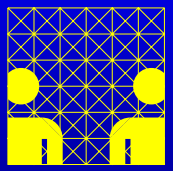
- “Generate & Test”-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid\; i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k,\; (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

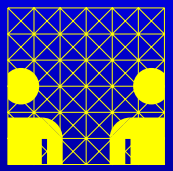
- “Generate & Test”-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

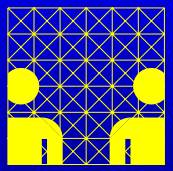
- "Generate & Test"-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$
  - $\neg\mathsf{SAT}(\mathfrak{A} \cup \neg\mathfrak{Q})$ w.r.t. $\mathfrak{T}$



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

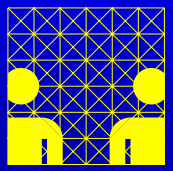
- "Generate & Test"-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$
  - $\neg\mathsf{SAT}(\mathfrak{A} \cup \neg\mathfrak{Q})$ w.r.t. $\mathfrak{T}$
  - $\neg\mathfrak{Q} =_{def} \neg q_1 \vee \neg q_2 \vee \cdots \vee \neg q_n$



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

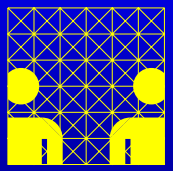
- "Generate & Test"-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$
  - $\neg\mathsf{SAT}(\mathfrak{A} \cup \neg\mathfrak{Q})$ w.r.t. $\mathfrak{T}$
  - $\neg\mathfrak{Q} =_{def} \neg q_1 \vee \neg q_2 \vee \cdots \vee \neg q_n$
  - $\neg\mathsf{SAT}(\mathfrak{A} \cup \neg\mathfrak{Q})$ iff $\forall i \in 1 \ldots n : \neg\mathsf{SAT}(\mathfrak{A} \cup \{\neg q_i\})$



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

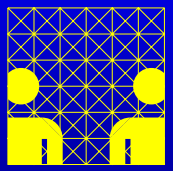
- “Generate & Test”-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$
- Algorithmus absolut ineffizient



# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

- “Generate & Test”-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$
- Algorithmus absolut ineffizient
- Konsistenz und Folgerbarkeit (Subsumption)

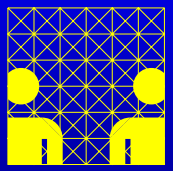


# (Zu) Primitiver Algorithmus

$$\{\,(i,j,k) \mid i,j,k \in \mathfrak{A}, i \neq j, i \neq k, j \neq k, \\ (\mathfrak{A},\mathfrak{T}) \models answer(x,y,z)_{x \leftarrow i, y \leftarrow j, z \leftarrow k}\,\}$$

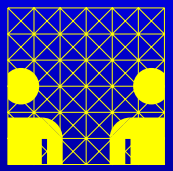
- "Generate & Test"-Algorithmus suggeriert
- Bilde alle $(i,j,k,\ldots)$-Tupel
- Substitutiere Query $\Rightarrow$ ABox $\mathfrak{Q}$
- Teste, ob $(\mathfrak{A},\mathfrak{T}) \models \mathfrak{Q}$
- Algorithmus absolut ineffizient
- Konsistenz und Folgerbarkeit (Subsumption)

$\Rightarrow$ Semantic Query Rewriting, Verwaltung und Wiederverwendung von gecachten Queries



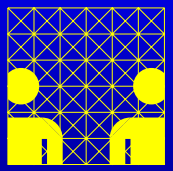
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz



# Verarbeitungs-Schritte

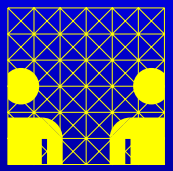
- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben



# Verarbeitungs-Schritte

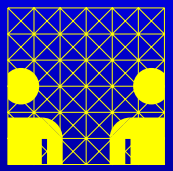
- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
  - Umschreiben in NNF (oder DNF), s. weiter unten
  - Anbieten einer benutzerfreundlicheren Syntax (“Macros”)
  - Hinzufügen von Hilfs-Konjunkten

```
(betty woman) ⇒
(and (bind-individual betty)
     (betty woman))
```



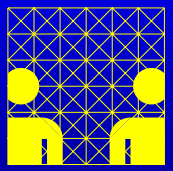
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing



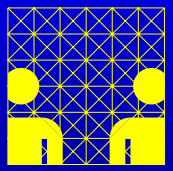
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
  - CLOS-Instanzen für
  - Variablen, “Nominals”,
  - Query-Konjunkte (Atome),
  - komplexe Queries (AND/OR, s. unten)



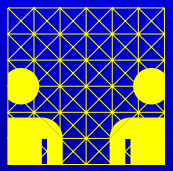
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben



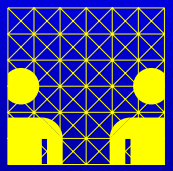
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
  - Verwende “Query Consistency”- und “Query Entailment”-Tests
  - Optimierung
  - Zusammenfassen von Konjunkten
  - Ersetzen von Konjunkten gegen Äquivalente
  - Entfernen/Hinzufügen von implizierten Konjunkten
  - berechne Cache-Referenzen



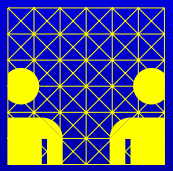
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung



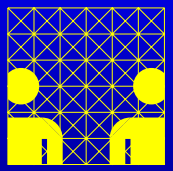
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung
  - simples kostenbasiertes Umordnen
  - verwende Cache-Referenzen



# Verarbeitungs-Schritte

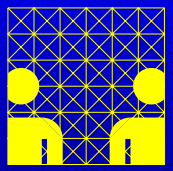
- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung
- Compilierung



# Verarbeitungs-Schritte

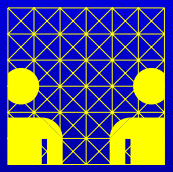
- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung
- Compilierung
  - Erzeuge für jede (Sub)query `query` ein LISP-Programm und binde es an den Slot `query-fn`
  - Rufe den LISP-Compiler auf: `(compile (query-fn query))`

  ⇒ Queries müssen nicht interpretiert werden!



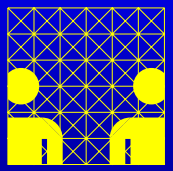
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung
- Compilierung
- Ausführung = Backtracking-Suche



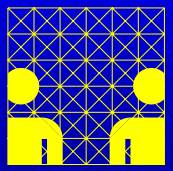
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung
- Compilierung
- Ausführung = Backtracking-Suche
  - `(funcall (query-fn query))`
  - das LISP-Programm ruft entsp. Retrieval-Operationen des RACER-Servers auf und navigiert durch die ABox (CSP)
  - Antworten werden gecached



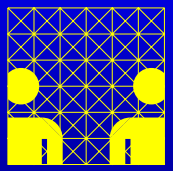
# Verarbeitungs-Schritte

- Vorweg: pragmatischer Ansatz
- Syntaktisches Umschreiben
- Parsing
- Semantisches Umschreiben
- Optimierung = Plan-Generierung
- Compilierung
- Ausführung = Backtracking-Suche



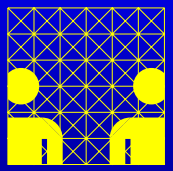
# ABox-Retrieval-Operationen

- Was kann zur Implementierung der “Conjunctive Queries” verwendet werden?



# ABox-Retrieval-Operationen

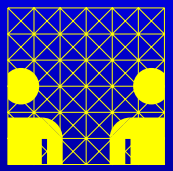
- Was kann zur Implementierung der “Conjunctive Queries” verwendet werden?
- Teuer (semantisch, Inferenz erforderlich):
  - `concept-instances (inherent)`
  - `related-individuals (⋆)`
  - `individual-fillers (⋆)`



# ABox-Retrieval-Operationen

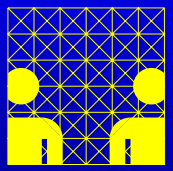
- Was kann zur Implementierung der “Conjunctive Queries” verwendet werden?
- Teuer (semantisch, Inferenz erforderlich):
  - `concept-instances (inherent)`
  - `related-individuals (`$\star$`)`
  - `individual-fillers (`$\star$`)`
- Billig (syntaktisch, keine Inferenz erforderlich):
  - `all-individuals`
  - `all-concept-assertions`
  - `all-role-assertions`
  - `all-role-ass.-for-ind.-in-domain`

  $\Rightarrow$ “Told Information”



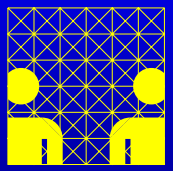
# Semantisches Umschreiben

- `(and (?x uncle) (?x professor))`
  $\Rightarrow$ `(?x (and uncle professor))`



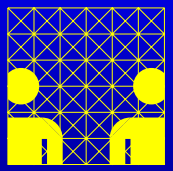
# Semantisches Umschreiben

- `(and (?x uncle) (?x professor))` $\Rightarrow$ `(?x (and uncle professor))`
- `(or (?x uncle) (?x professor))` $\not\Rightarrow$ `(?x (or uncle professor))` !



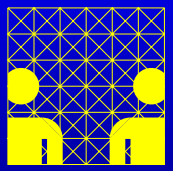
# Semantisches Umschreiben

- (and (?x uncle) (?x professor)) $\Rightarrow$ (?x (and uncle professor))
- (or (?x uncle) (?x professor)) $\not\Rightarrow$ (?x (or uncle professor)) !
- (and (?x woman) (?x mother)) $\Rightarrow$ (?x mother)



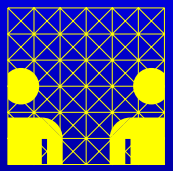
# Semantisches Umschreiben

- (and (?x uncle) (?x professor)) $\Rightarrow$ (?x (and uncle professor))
- (or (?x uncle) (?x professor)) $\not\Rightarrow$ (?x (or uncle professor)) !
- (and (?x woman) (?x mother)) $\Rightarrow$ (?x mother)
- (and (?x man) (?x woman)) $\Rightarrow$ (?x bottom) $\Rightarrow$ NIL



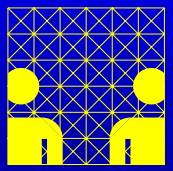
# Semantisches Umschreiben

- (and (?x uncle) (?x professor)) $\Rightarrow$ (?x (and uncle professor))
- (or (?x uncle) (?x professor)) $\not\Rightarrow$ (?x (or uncle professor)) !
- (and (?x woman) (?x mother)) $\Rightarrow$ (?x mother)
- (and (?x man) (?x woman)) $\Rightarrow$ (?x bottom) $\Rightarrow$ NIL
- (and (?x woman) (?x ?y h-child)) $\Rightarrow$ (and (?x mother) (?y human) ...)



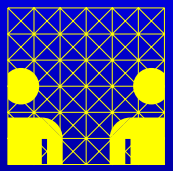
# Semantisches Umschreiben

- (and (?x uncle) (?x professor)) $\Rightarrow$ (?x (and uncle professor))
- (or (?x uncle) (?x professor)) $\not\Rightarrow$ (?x (or uncle professor)) !
- (and (?x woman) (?x mother)) $\Rightarrow$ (?x mother)
- (and (?x man) (?x woman)) $\Rightarrow$ (?x bottom) $\Rightarrow$ NIL
- (and (?x woman) (?x ?y h-child)) $\Rightarrow$ (and (?x mother) (?y human) ...)
- (and (?x ?y has-child) (?y man)) $\Rightarrow$ (and (?y man) (?y ?x child-of) ...)



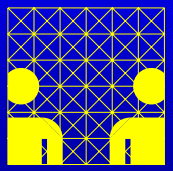
# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`



# Heuristische Optimierung

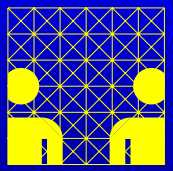
- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne



# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

⊖ `(?x woman) (?y man) (?x ?y has-child)`

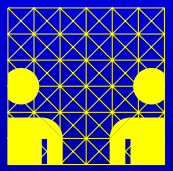


# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

⊖ `(?x woman) (?y man) (?x ?y has-child)`

⊕ `(?x woman) (?x ?y has-child) (?y man)`



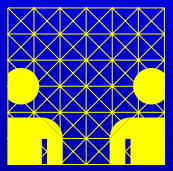
# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

⊖ `(?x woman) (?y man) (?x ?y has-child)`

⊕ `(?x woman) (?x ?y has-child) (?y man)`

⊕⊕ `(?y man) (?y ?x child-of) (?x woman)`



# Heuristische Optimierung

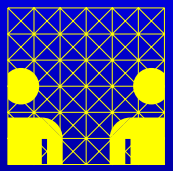
- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

$\ominus$ `(?x woman) (?y man) (?x ?y has-child)`

$\oplus$ `(?x woman) (?x ?y has-child) (?y man)`

$\oplus\oplus$ `(?y man) (?y ?x child-of) (?x woman)`

$\Rightarrow$ nutze heuristische Suche, um einen “guten” Plan zu finden (suboptimal, wegen $n!$-Aufwand)



# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

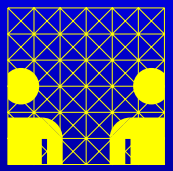
$\ominus$ `(?x woman) (?y man) (?x ?y has-child)`

$\oplus$ `(?x woman) (?x ?y has-child) (?y man)`

$\oplus\oplus$ `(?y man) (?y ?x child-of) (?x woman)`

$\Rightarrow$ nutze heuristische Suche, um einen “guten” Plan zu finden (suboptimal, wegen $n!$-Aufwand)

- Ziel: Minimierung des Branching-Factors im Suchbaum



# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

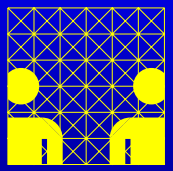
$\ominus$ `(?x woman) (?y man) (?x ?y has-child)`

$\oplus$ `(?x woman) (?x ?y has-child) (?y man)`

$\oplus\oplus$ `(?y man) (?y ?x child-of) (?x woman)`

$\Rightarrow$ nutze heuristische Suche, um einen "guten" Plan zu finden (suboptimal, wegen $n!$-Aufwand)

- Ziel: Minimierung des Branching-Factors im Suchbaum
- Heuristic: "Most Constrained (Generator) First"



# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

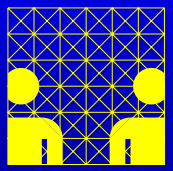
$\ominus$ `(?x woman) (?y man) (?x ?y has-child)`

$\oplus$ `(?x woman) (?x ?y has-child) (?y man)`

$\oplus\oplus$ `(?y man) (?y ?x child-of) (?x woman)`

$\Rightarrow$ nutze heuristische Suche, um einen “guten” Plan zu finden (suboptimal, wegen $n!$-Aufwand)

- Ziel: Minimierung des Branching-Factors im Suchbaum
- Heuristic: “Most Constrained (Generator) First”
- verwende ABox-Statistiken wenn verfügbar



# Heuristische Optimierung

- `(?x woman) (?y man) (?x ?y has-child)`
- $3! = 6$ verschiedene Abarbeitungs-Pläne

$\ominus$ `(?x woman) (?y man) (?x ?y has-child)`

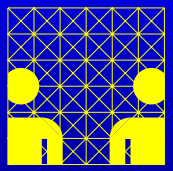
$\oplus$ `(?x woman) (?x ?y has-child) (?y man)`

$\oplus\oplus$ `(?y man) (?y ?x child-of) (?x woman)`

$\Rightarrow$ nutze heuristische Suche, um einen "guten" Plan zu finden (suboptimal, wegen $n!$-Aufwand)

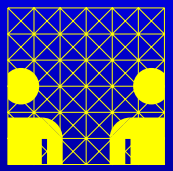
- Ziel: Minimierung des Branching-Factors im Suchbaum
- Heuristic: "Most Constrained (Generator) First"
- verwende ABox-Statistiken wenn verfügbar

? `(?x new-concept1) (?y new-concept2)`



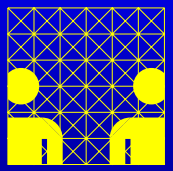
# Query Repository

- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)



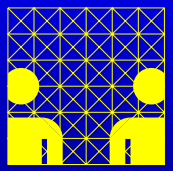
# Query Repository

- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)
- Repository = Query-Taxonomie, Ergebnis-Tupelmengen



# Query Repository

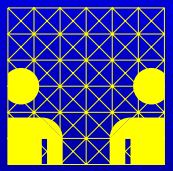
- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)
- Repository = Query-Taxonomie, Ergebnis-Tupelmengen
- Klassifiziere neue Query in Query-Taxonomie ein
- nutze Taxonomie: verwende exact oder superset Cache-Eintrag



# Query Repository

- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)
- Repository = Query-Taxonomie, Ergebnis-Tupelmengen
- Klassifiziere neue Query in Query-Taxonomie ein
- nutze Taxonomie: verwende exact oder superset Cache-Eintrag

⊕ aufwendige Laufzeit-Suche kann gegen Lookups ersetzt werden

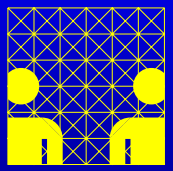


# Query Repository

- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)
- Repository = Query-Taxonomie, Ergebnis-Tupelmengen
- Klassifiziere neue Query in Query-Taxonomie ein
- nutze Taxonomie: verwende exact oder superset Cache-Eintrag

⊕ aufwendige Laufzeit-Suche kann gegen Lookups ersetzt werden

⊖ Speicherbedarf, Verwaltung der Taxonomie



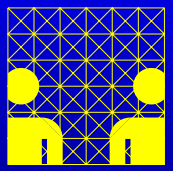
# Query Repository

- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)
- Repository = Query-Taxonomie, Ergebnis-Tupelmengen
- Klassifiziere neue Query in Query-Taxonomie ein
- nutze Taxonomie: verwende exact oder superset Cache-Eintrag

⊕ aufwendige Laufzeit-Suche kann gegen Lookups ersetzt werden

⊖ Speicherbedarf, Verwaltung der Taxonomie

⊖ höherer Zeitbedarf zur Compilezeit



# Query Repository

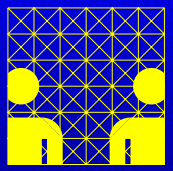
- Idee: cache Ergebnis-Tupelmengen von Queries und wiederverwende sie (optional)
- Repository = Query-Taxonomie, Ergebnis-Tupelmengen
- Klassifiziere neue Query in Query-Taxonomie ein
- nutze Taxonomie: verwende exact oder superset Cache-Eintrag

⊕ aufwendige Laufzeit-Suche kann gegen Lookups ersetzt werden

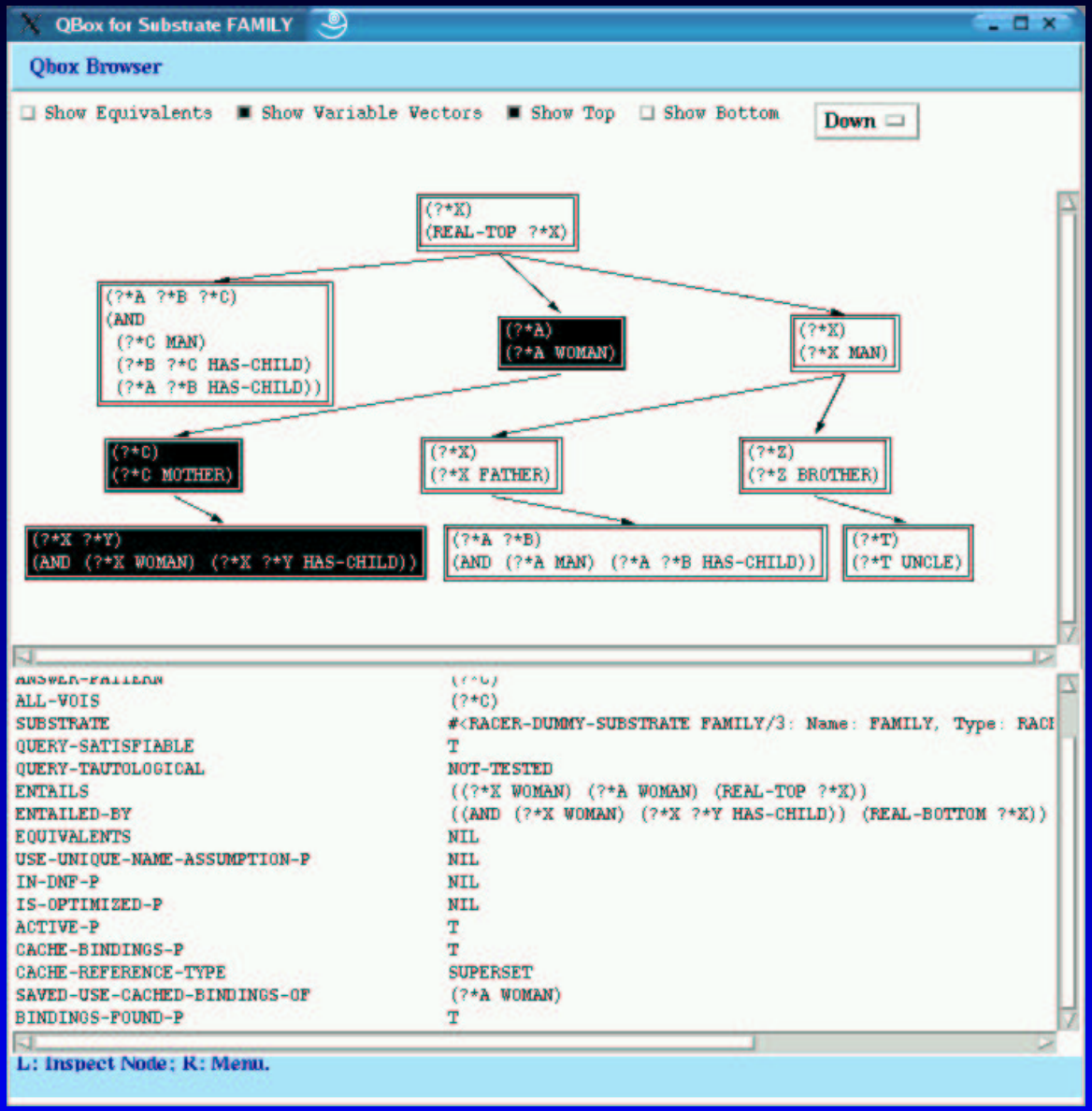
⊖ Speicherbedarf, Verwaltung der Taxonomie

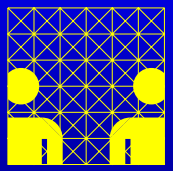
⊖ höherer Zeitbedarf zur Compilezeit

⇒ lohnt nur bei Queries m. hohem Laufzeitaufwand



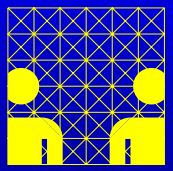
# Query Repository





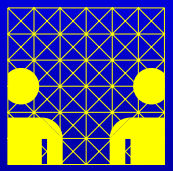
# Negation für Queries

- Open World Assumption



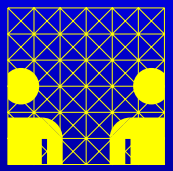
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`



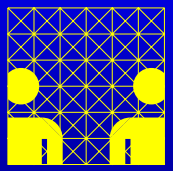
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`
  - `(?x (not grandfather))` $\Rightarrow$ `NIL`



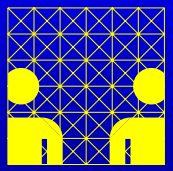
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`
  - `(?x (not grandfather))` $\Rightarrow$ `NIL`
  - PROLOG:
    `(charles alice betty eve doris)`



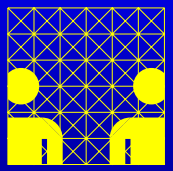
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`
  - `(?x (not grandfather))` $\Rightarrow$ `NIL`
  - PROLOG:
    `(charles alice betty eve doris)`
- Manchmal möchte man “Negation As Failure” bzw. “Closed Word Assumption” = “was ich nicht beweisen kann, wird als falsch angenommen”



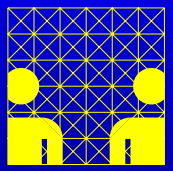
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`
  - `(?x (not grandfather))` $\Rightarrow$ `NIL`
  - PROLOG:
    `(charles alice betty eve doris)`
- Manchmal möchte man “Negation As Failure” bzw. “Closed Word Assumption” = “was ich nicht beweisen kann, wird als falsch angenommen”
- Neue zusätzliche Syntax:



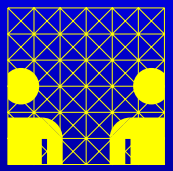
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`
  - `(?x (not grandfather))` $\Rightarrow$ `NIL`
  - PROLOG:
    `(charles alice betty eve doris)`
- Manchmal möchte man “Negation As Failure” bzw. “Closed Word Assumption” = “was ich nicht beweisen kann, wird als falsch angenommen”
- Neue zusätzliche Syntax:
  - unäre Atome:
    `(not (?x grandfather))`



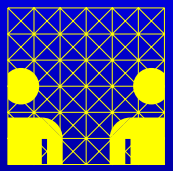
# Negation für Queries

- Open World Assumption
  - `(?x grandfather)` $\Rightarrow$ `NIL`
  - `(?x (not grandfather))` $\Rightarrow$ `NIL`
  - PROLOG:
    `(charles alice betty eve doris)`
- Manchmal möchte man “Negation As Failure” bzw. “Closed Word Assumption” = “was ich nicht beweisen kann, wird als falsch angenommen”
- Neue zusätzliche Syntax:
  - unäre Atome:
    `(not (?x grandfather))`
  - binäre Atome:
    `(not (?x ?y has-child))`



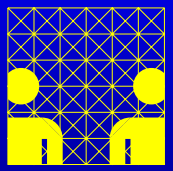
# Semantik negierter Atome

- Unäre Atome



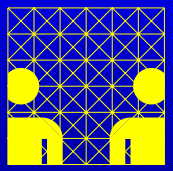
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`



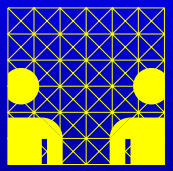
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{\, i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \,\}$



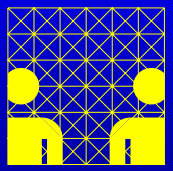
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{\, i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \,\}$
  - individuals($\mathfrak{A}$) $\setminus$ answer(`(?x C)`)



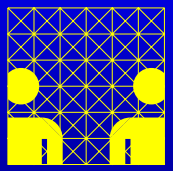
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{\, i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \,\}$
  - $\mathsf{individuals}(\mathfrak{A}) \setminus \mathsf{answer}(\texttt{(?x C)})$
- Binäre Atome



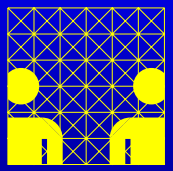
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{ i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \}$
  - $\mathsf{individuals}(\mathfrak{A}) \setminus \mathsf{answer}(\texttt{(?x C)})$
- Binäre Atome
  - `(not (?x ?y R))`



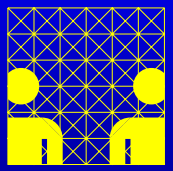
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{\, i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \,\}$
  - individuals($\mathfrak{A}$) \ answer(`(?x C)`)
- Binäre Atome
  - `(not (?x ?y R))`
  - $\{\, (i,j) \mid i,j \in \mathfrak{A}, i \neq j,$
    $(\mathfrak{A}, \mathfrak{T}) \not\models answer(x,y)_{x \leftarrow i, y \leftarrow j} \,\}$



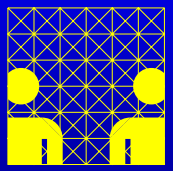
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{\, i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \,\}$
  - $\mathsf{individuals}(\mathfrak{A}) \setminus \mathsf{answer}(\texttt{(?x C)})$
- Binäre Atome
  - `(not (?x ?y R))`
  - $\{\, (i,j) \mid i,j \in \mathfrak{A}, i \neq j, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x,y)_{x \leftarrow i, y \leftarrow j} \,\}$
  - $(\mathsf{individuals}(\mathfrak{A}) \times \mathsf{individuals}(\mathfrak{A})) \setminus \mathsf{answer}(\texttt{(?x ?y R)})$



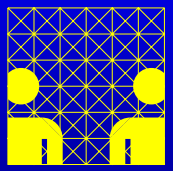
# Semantik negierter Atome

- Unäre Atome
  - `(not (?x C))`
  - $\{\, i \mid i \in \mathfrak{A}, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x)_{x \leftarrow i} \,\}$
  - $\mathsf{individuals}(\mathfrak{A}) \setminus \mathsf{answer}(\texttt{(?x C)})$
- Binäre Atome
  - `(not (?x ?y R))`
  - $\{\, (i,j) \mid i,j \in \mathfrak{A}, i \neq j, (\mathfrak{A}, \mathfrak{T}) \not\models answer(x,y)_{x \leftarrow i, y \leftarrow j} \,\}$
  - $(\mathsf{individuals}(\mathfrak{A}) \times \mathsf{individuals}(\mathfrak{A})) \setminus \mathsf{answer}(\texttt{(?x ?y R)})$
- Reasoning mit dieses Atomen? $\Rightarrow$ “future research”



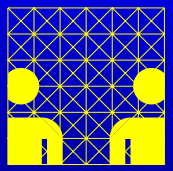
# Negierte Nominals?

- `(betty woman)`



# Negierte Nominals?

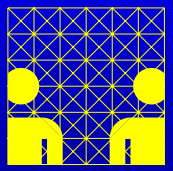
- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$



# Negierte Nominals?

- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$

$=$ (and (bind-individual betty) (betty woman))

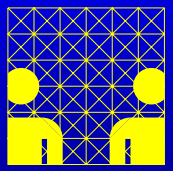


# Negierte Nominals?

- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$

$=$ (and (bind-individual betty) (betty woman))

$\Rightarrow$ (not (and (bind-individual betty) (betty woman)))



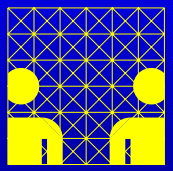
# Negierte Nominals?

- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$

$=$ (and (bind-individual betty) (betty woman))

$\Rightarrow$ (not (and (bind-individual betty) (betty woman)))

$=$ (or (not (bind-individual betty)) (not (betty woman)))



# Negierte Nominals?

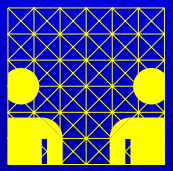
- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$

$=$ (and (bind-individual betty) (betty woman))

$\Rightarrow$ (not (and (bind-individual betty) (betty woman)))

$=$ (or (not (bind-individual betty)) (not (betty woman)))

$=$ $answer(betty) =_{def} \; betty \neq betty\_constant \vee \neg woman(betty)$



# Negierte Nominals?

- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$

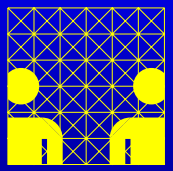
$=$ (and (bind-individual betty) (betty woman))

$\Rightarrow$ (not (and (bind-individual betty) (betty woman)))

$=$ (or (not (bind-individual betty)) (not (betty woman)))

$=$ $answer(betty) =_{def} \; betty \neq betty\_constant \vee \neg woman(betty)$

$\Rightarrow$ aus negierten “Nominals” werden Variablen



# Negierte Nominals?

- `(betty woman)`
- $answer(betty) =_{def} \; betty = betty\_constant \wedge woman(betty)$

$=$ (and (bind-individual betty) (betty woman))

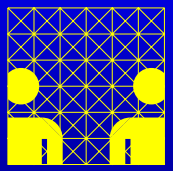
$\Rightarrow$ (not (and (bind-individual betty) (betty woman)))

$=$ (or (not (bind-individual betty)) (not (betty woman)))

$=$ $answer(betty) =_{def} \; betty \neq betty\_constant \vee \neg woman(betty)$

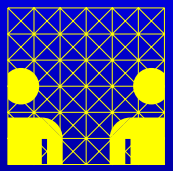
$\Rightarrow$ aus negierten “Nominals” werden Variablen

- $(\mathsf{answer}((?\mathrm{x}\ \mathrm{C})) \cup \mathsf{answer}((\mathrm{not}\ (?\mathrm{x}\ \mathrm{C})))) = \mathsf{answer}(?\mathrm{x}\ \mathrm{TOP}) = \mathsf{individuals}(\mathfrak{A})$



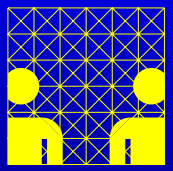
# Disjunktive Queries

- ebenfalls wünschenswert



# Disjunktive Queries

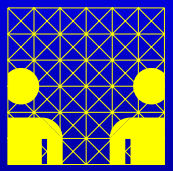
- ebenfalls wünschenswert
- zusätzliche Syntax: `OR`



# Disjunktive Queries

- ebenfalls wünschenswert
- zusätzliche Syntax: `OR`

$\Rightarrow$ Disjunctive Conjunctive Queries (mit mglw. negierten Atomen)

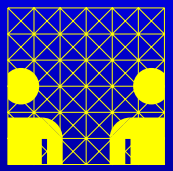


# Disjunktive Queries

- ebenfalls wünschenswert
- zusätzliche Syntax: `OR`

$\Rightarrow$ Disjunctive Conjunctive Queries (mit mglw. negierten Atomen)

- jede Query kann in Negations-Normalform gebracht werden (NNF = alle negierten Queries sind Atome)



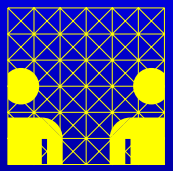
# Disjunktive Queries

- ebenfalls wünschenswert
- zusätzliche Syntax: `OR`

⇒ Disjunctive Conjunctive Queries (mit mglw. negierten Atomen)

- jede Query kann in Negations-Normalform gebracht werden (NNF = alle negierten Queries sind Atome)

⇒ Beliebige “Boolean Queries” können verarbeitet werden



# Disjunktive Queries

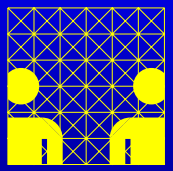
- ebenfalls wünschenswert
- zusätzliche Syntax: `OR`

⇒ Disjunctive Conjunctive Queries (mit mglw. negierten Atomen)

- jede Query kann in Negations-Normalform gebracht werden (NNF = alle negierten Queries sind Atome)

⇒ Beliebige “Boolean Queries” können verarbeitet werden

- aber: was soll z.B. `(or (?x C) (?y D))` genau bedeuten (Stelligkeit?)



# Disjunktive Queries

- ebenfalls wünschenswert
- zusätzliche Syntax: `OR`

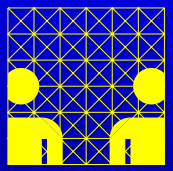
$\Rightarrow$ Disjunctive Conjunctive Queries (mit mglw. negierten Atomen)

- jede Query kann in Negations-Normalform gebracht werden (NNF = alle negierten Queries sind Atome)

$\Rightarrow$ Beliebige "Boolean Queries" können verarbeitet werden

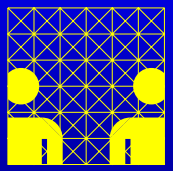
- aber: was soll z.B. `(or (?x C) (?y D))` genau bedeuten (Stelligkeit?)

$=$
```
(not (and (not (?x C))
          (not (?y D))))
```



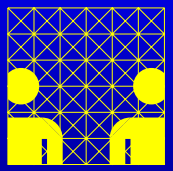
# Semantik des OR

- Bsp. `(or (?x C) (?y D))`



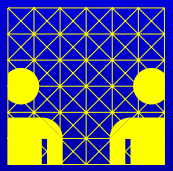
# Semantik des OR

- Bsp. `(or (?x C) (?y D))`
- Forderung: DeMorgan soll gelten



# Semantik des `OR`

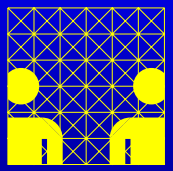
- Bsp. `(or (?x C) (?y D))`
- Forderung: DeMorgan soll gelten
- Semantik `OR` = Vereinigung der Tupelmengen der Disjunkte



# Semantik des `OR`

- Bsp. `(or (?x C) (?y D))`
- Forderung: DeMorgan soll gelten
- Semantik `OR` = Vereinigung der Tupelmengen der Disjunkte

$\Rightarrow$ nur bei gleicher Stelligkeit wohldefiniert



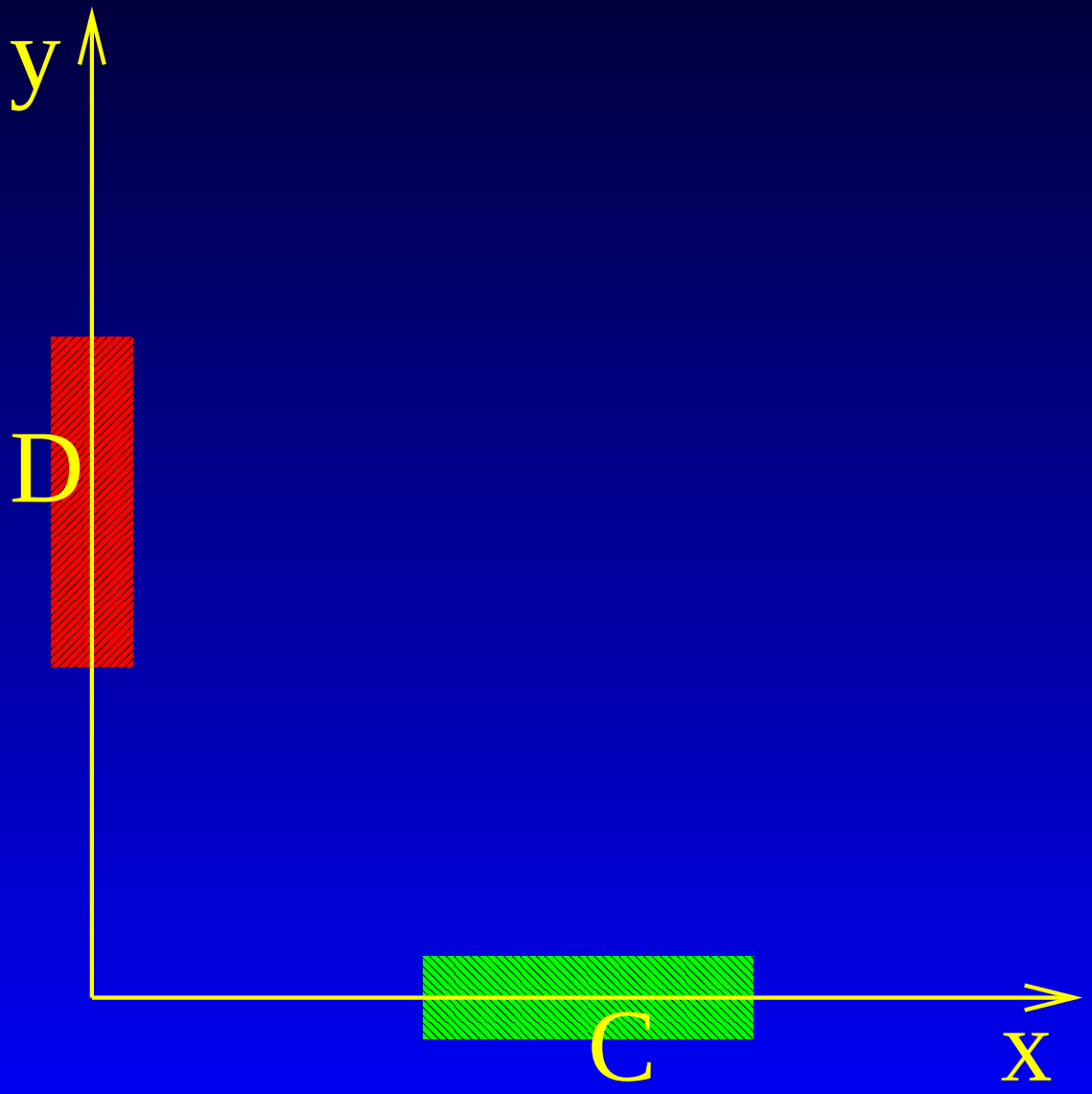
# Semantik des `OR`

- Bsp. `(or (?x C) (?y D))`
- Forderung: DeMorgan soll gelten
- Semantik `OR` = Vereinigung der Tupelmengen der Disjunkte

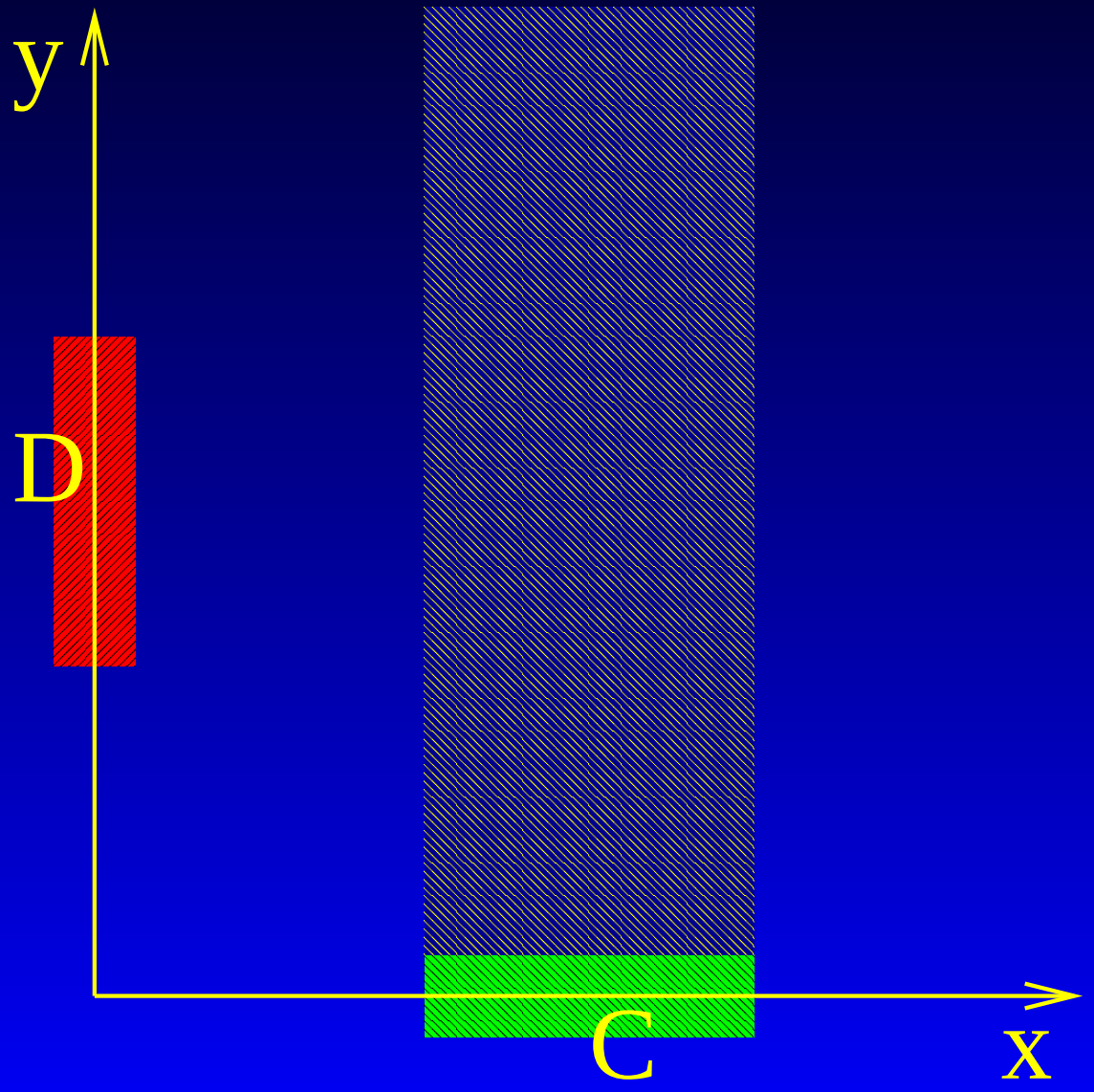
$\Rightarrow$ nur bei gleicher Stelligkeit wohldefiniert

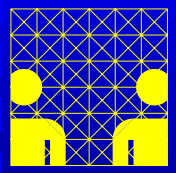
- Beim `AND` ist es klar: jede unterschiedlich benannte Variable spannt eine Achse im $n$-dimensionalen Kreuzprodukt-Raum auf

# Semantik des OR

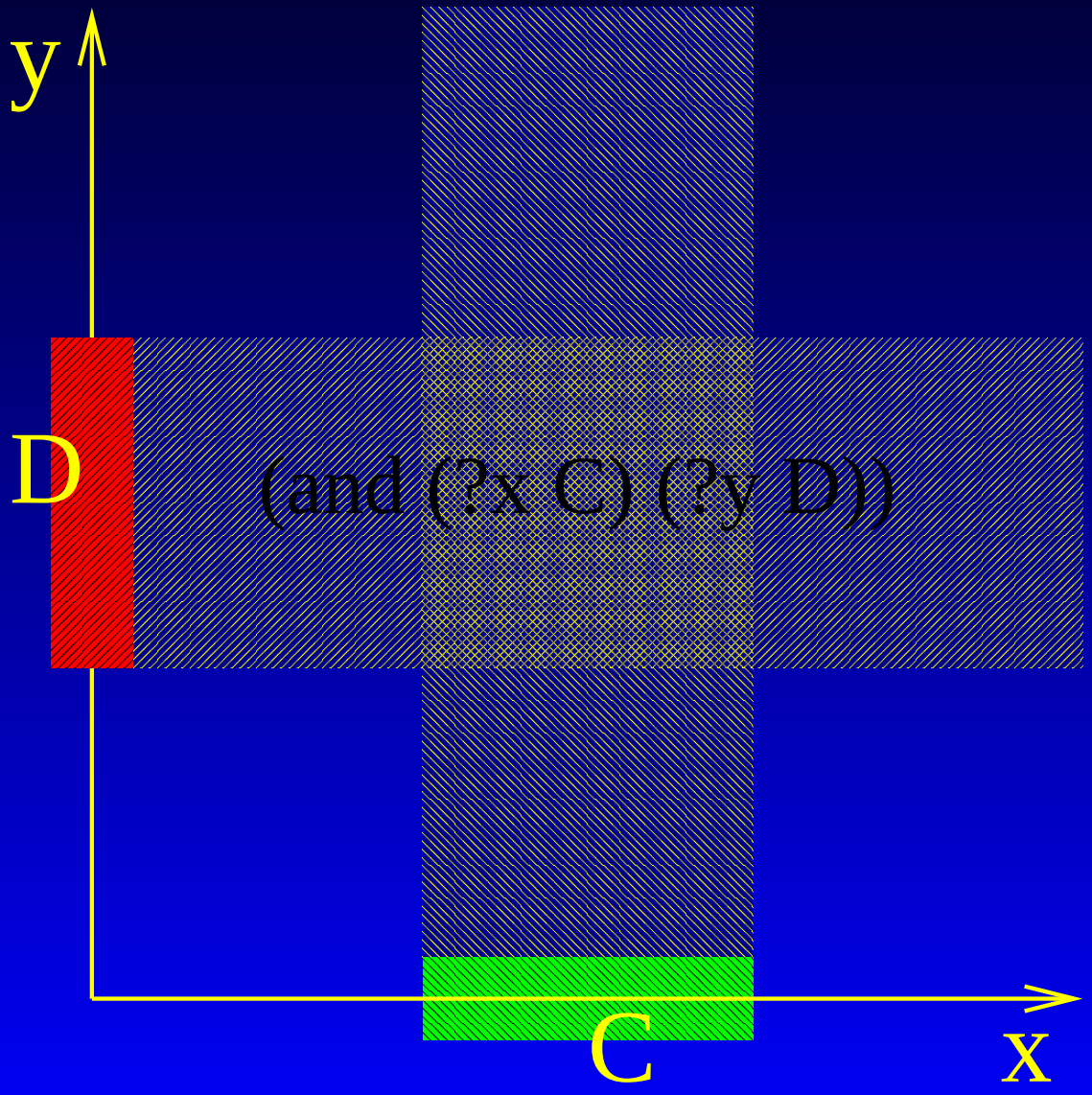


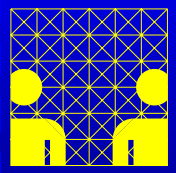
# Semantik des OR



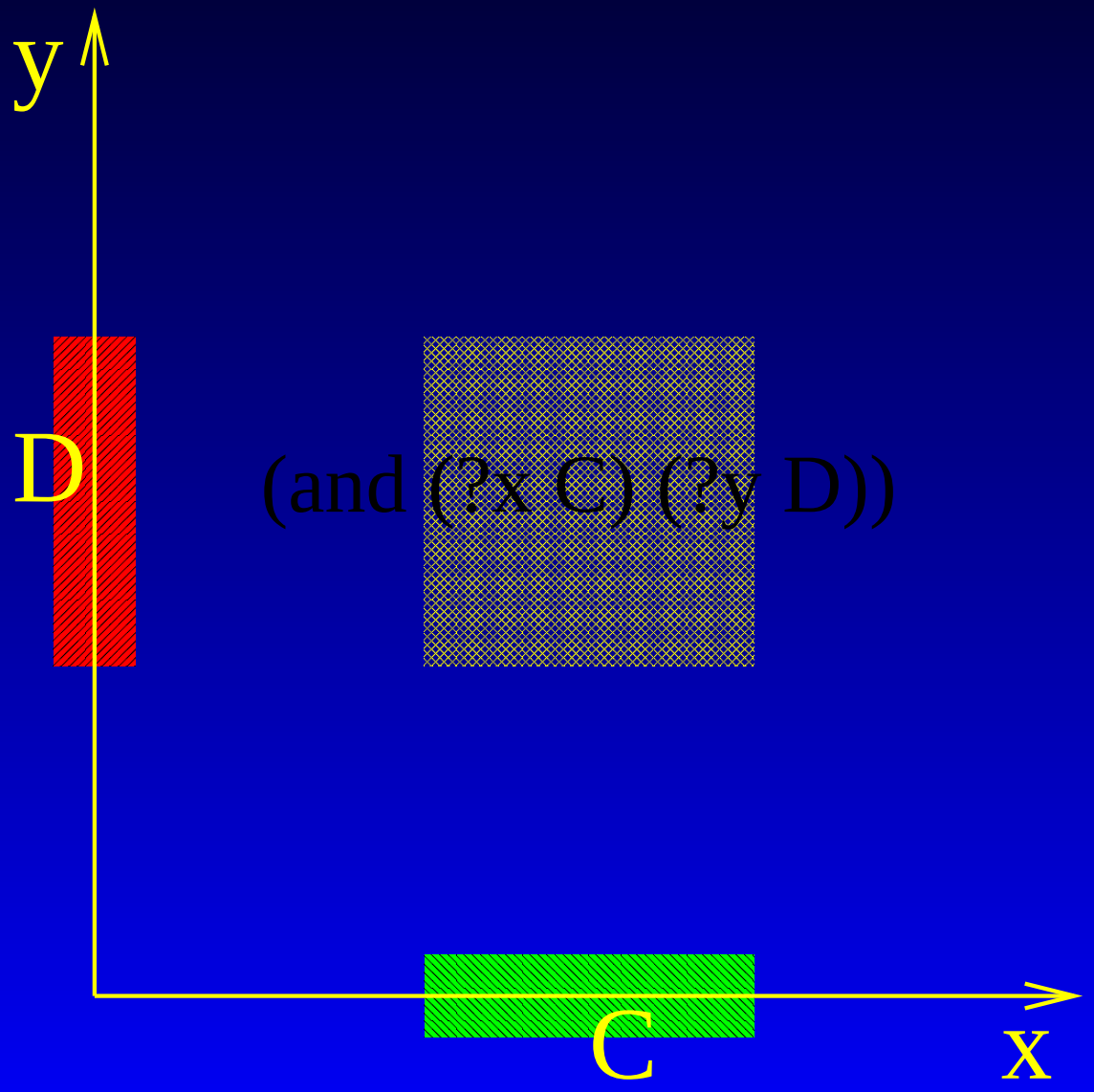


# Semantik des OR

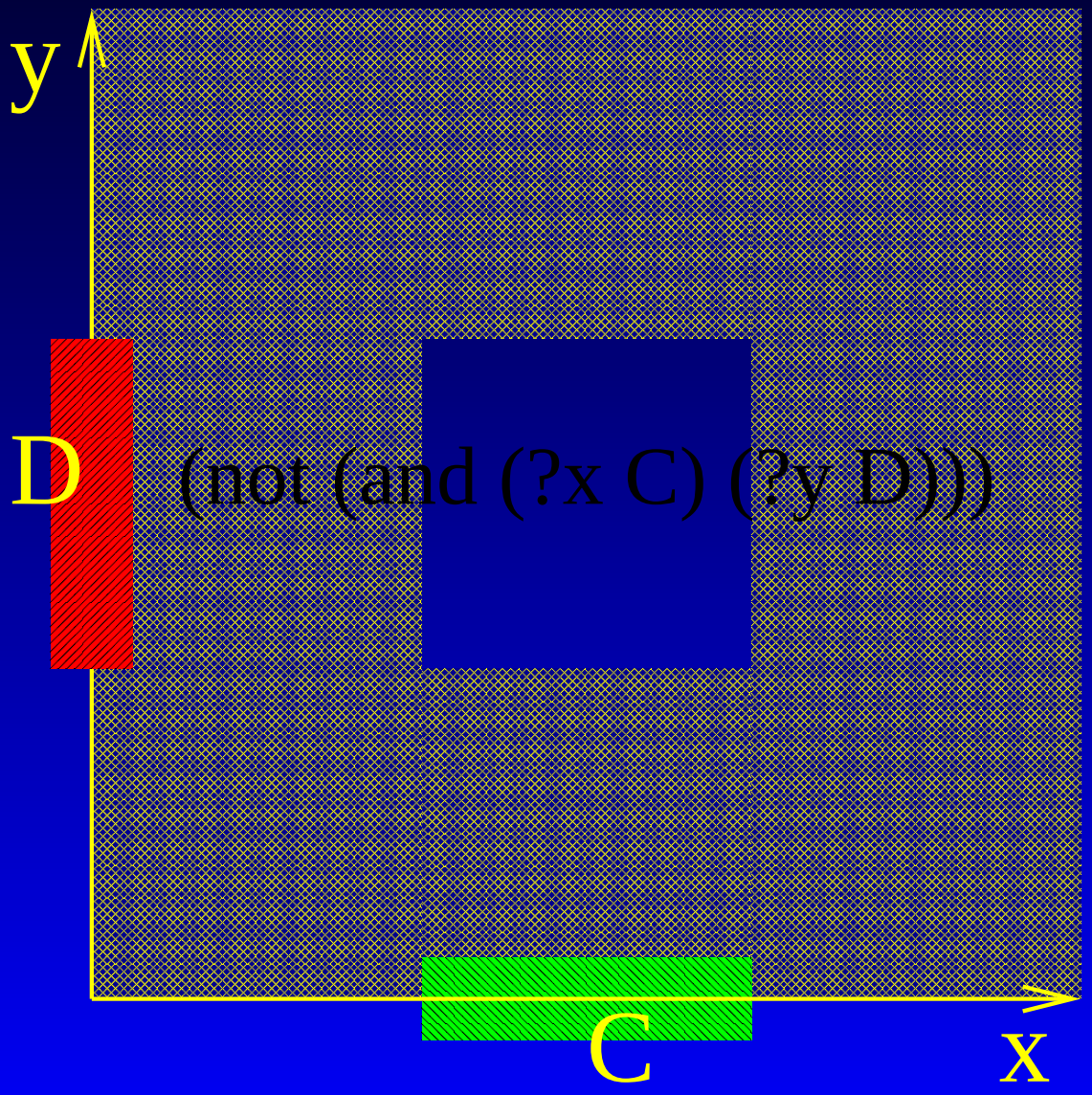




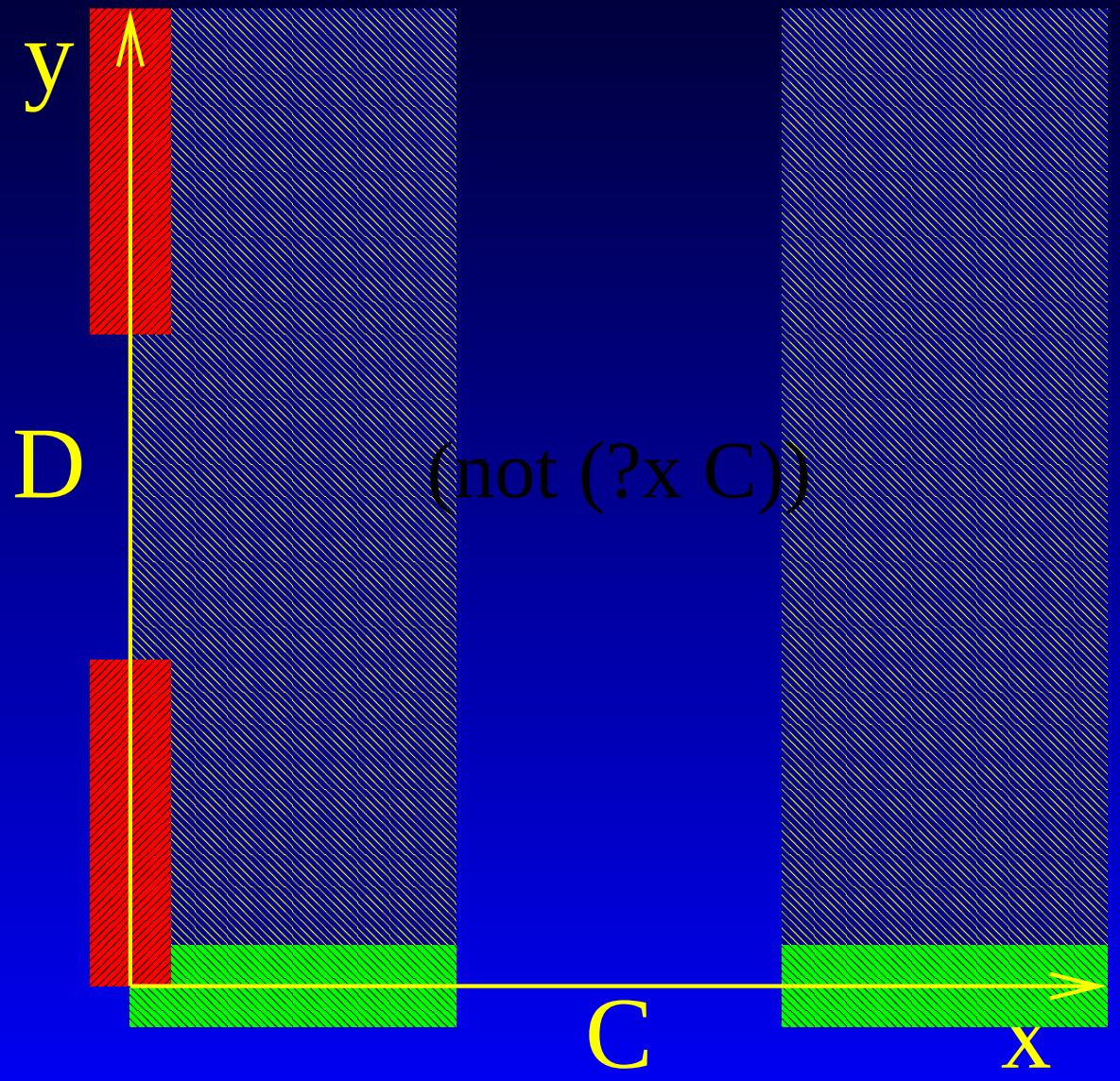
# Semantik des OR

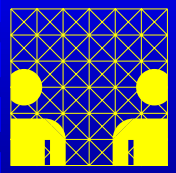


# Semantik des OR

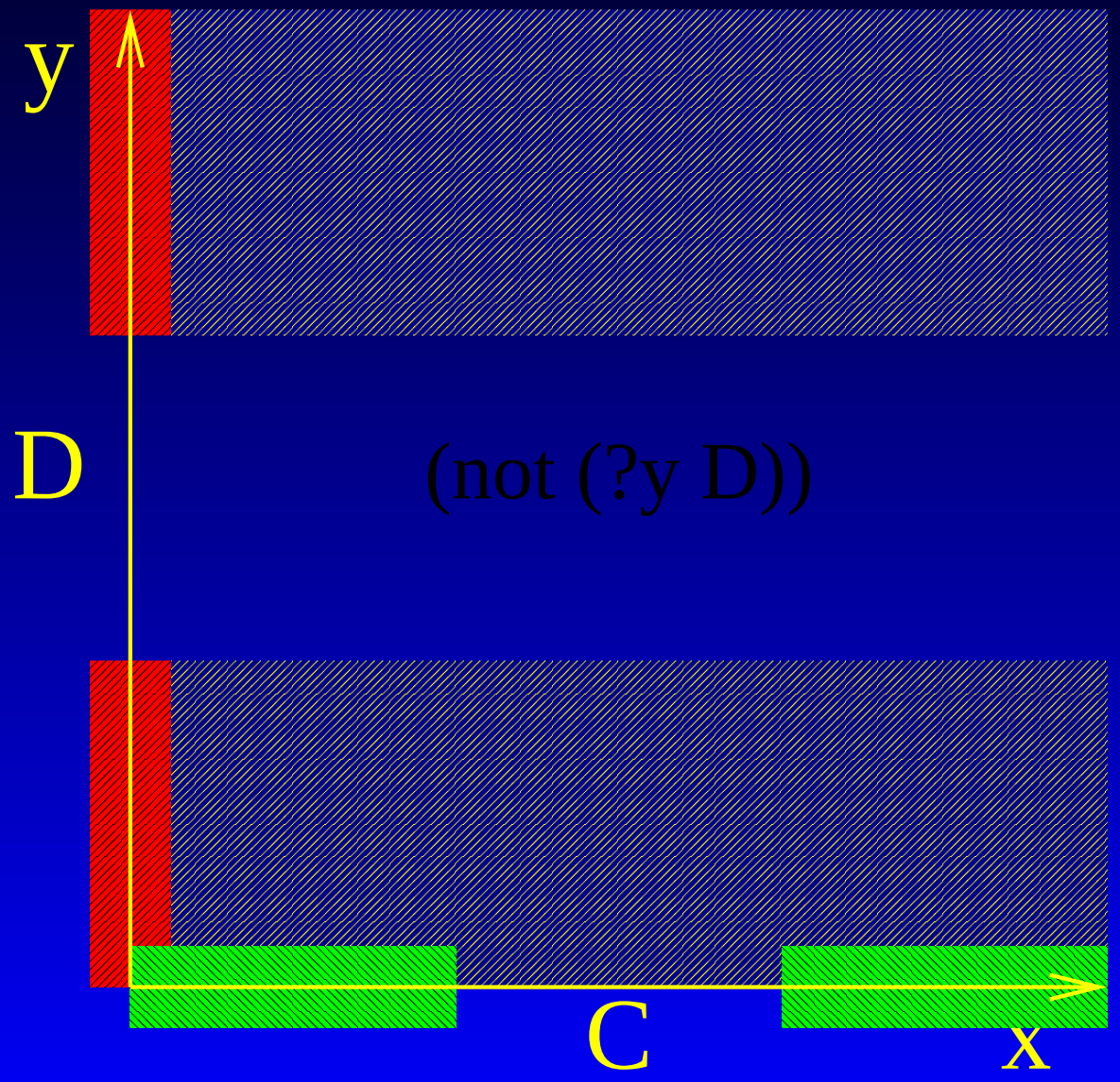


# Semantik des OR

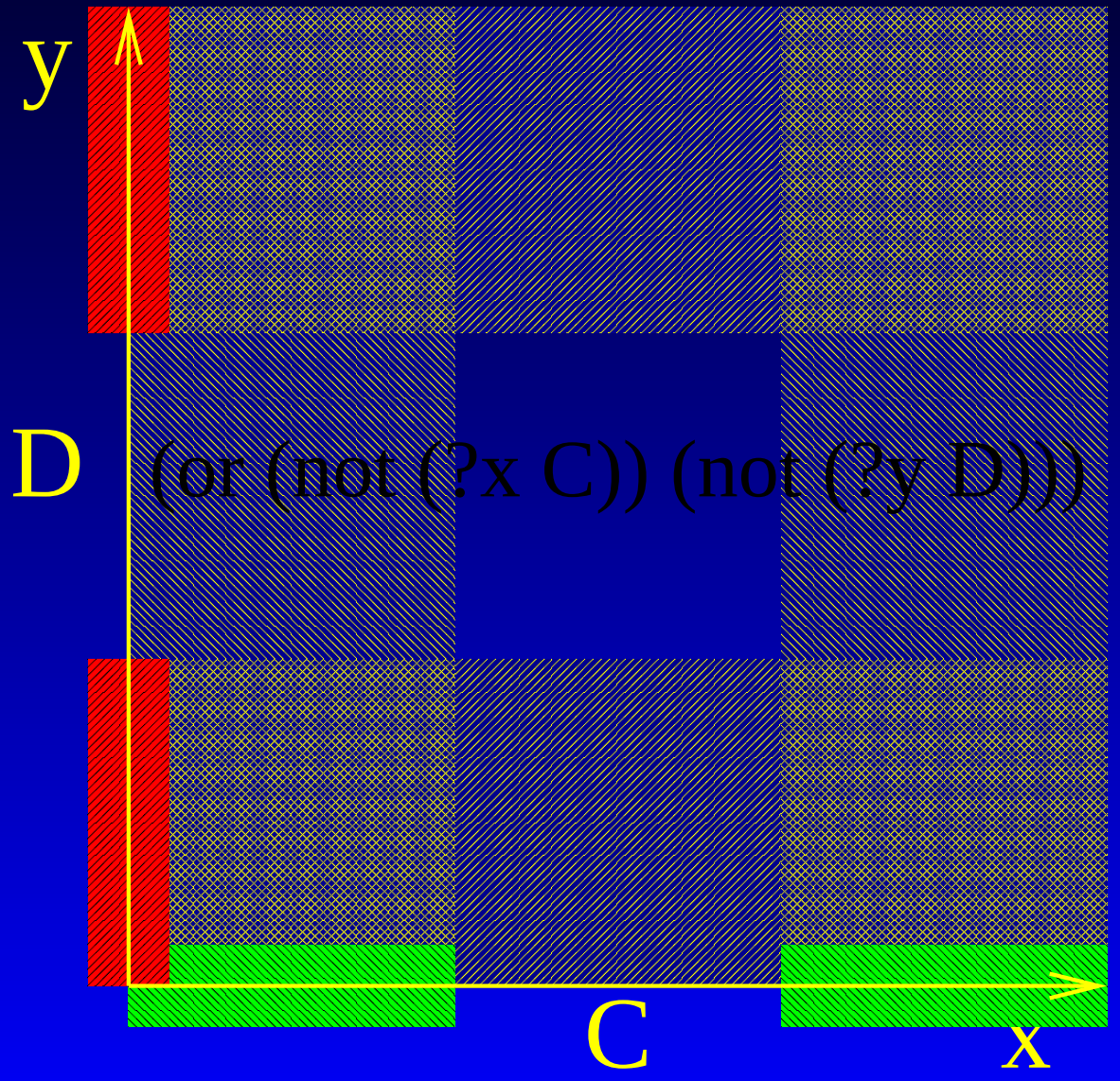


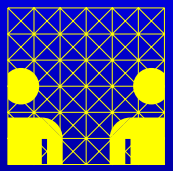


# Semantik des OR



# Semantik des OR





# Semantik des OR

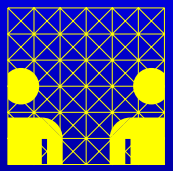
- Bsp. `(or (?x C) (?y D))`
- Forderung: DeMorgan soll gelten
- Semantik `OR` = Vereinigung der Tupelmengen der Disjunkte

$\Rightarrow$ nur bei gleicher Stelligkeit wohldefiniert

- Beim `AND` ist es klar: jede unterschiedlich benannte Variable spannt eine Achse im $n$-dimensionalen Kreuzprodukt-Raum auf

$\Rightarrow$ `(or (?x C) (?y D))` $\Rightarrow$

```
(or (and (?x C) (?y TOP))
    (and (?x TOP) (?y D)))
```



# Semantik des OR

- Bsp. `(or (?x C) (?y D))`
- Forderung: DeMorgan soll gelten
- Semantik `OR` = Vereinigung der Tupelmengen der Disjunkte

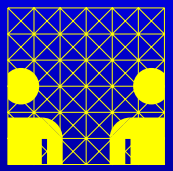
$\Rightarrow$ nur bei gleicher Stelligkeit wohldefiniert

- Beim `AND` ist es klar: jede unterschiedlich benannte Variable spannt eine Achse im $n$-dimensionalen Kreuzprodukt-Raum auf

$\Rightarrow$ `(or (?x C) (?y D))` $\Rightarrow$

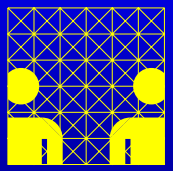
```
(or (and (?x C) (?y TOP))
    (and (?x TOP) (?y D)))
```

$\Rightarrow$ alle Disjunkte verwenden gleiche Variablen



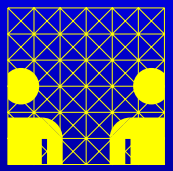
# Verarbeitung beliebiger Queries

- Bringe Query in DNF



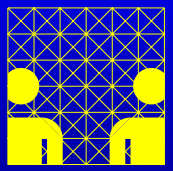
# Verarbeitung beliebiger Queries

- Bringe Query in DNF
- Sorge beim syntaktischen Umschreiben dafür, dass alle Disjunkte die gleichen Variablen referenzieren



# Verarbeitung beliebiger Queries

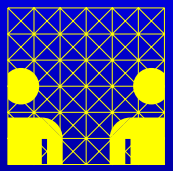
- Bringe Query in DNF
- Sorge beim syntaktischen Umschreiben dafür, dass alle Disjunkte die gleichen Variablen referenzieren
  - Addition von `(?y TOP)`-Konjunkten



# Verarbeitung beliebiger Queries

- Bringe Query in DNF
- Sorge beim syntaktischen Umschreiben dafür, dass alle Disjunkte die gleichen Variablen referenzieren
  - Addition von `(?y TOP)`-Konjunkten

$\Rightarrow$ alle Disjunkte referenzieren die gleiche Variablen

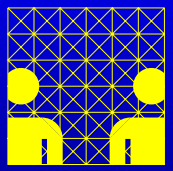


# Verarbeitung beliebiger Queries

- Bringe Query in DNF
- Sorge beim syntaktischen Umschreiben dafür, dass alle Disjunkte die gleichen Variablen referenzieren
  - Addition von `(?y TOP)`-Konjunkten

$\Rightarrow$ alle Disjunkte referenzieren die gleiche Variablen

- jedes Disjunkt (weil in DNF ist das ein Atom oder ein Und) wird unabhängig optimiert



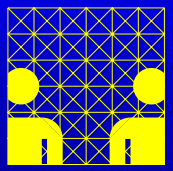
# Verarbeitung beliebiger Queries

- Bringe Query in DNF
- Sorge beim syntaktischen Umschreiben dafür, dass alle Disjunkte die gleichen Variablen referenzieren
  - Addition von `(?y TOP)`-Konjunkten

$\Rightarrow$ alle Disjunkte referenzieren die gleiche Variablen

- jedes Disjunkt (weil in DNF ist das ein Atom oder ein Und) wird unabhängig optimiert

$\oplus$ einfach implementierbar



# Verarbeitung beliebiger Queries

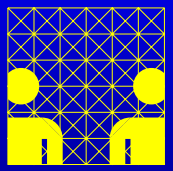
- Bringe Query in DNF
- Sorge beim syntaktischen Umschreiben dafür, dass alle Disjunkte die gleichen Variablen referenzieren
  - Addition von `(?y TOP)`-Konjunkten

$\Rightarrow$ alle Disjunkte referenzieren die gleiche Variablen

- jedes Disjunkt (weil in DNF ist das ein Atom oder ein Und) wird unabhängig optimiert

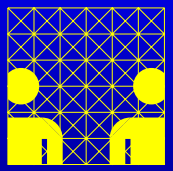
$\oplus$ einfach implementierbar

$\ominus$ DNF-Query kann exponentiell viel größer sein als die originale Query



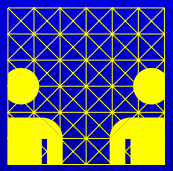
# Ausblick

- Haben:



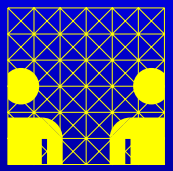
# Ausblick

- Haben:
  - Einfache, aber interessante RACER-”Erweiterung”



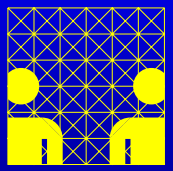
# Ausblick

- Haben:
  - Einfache, aber interessante RACER-”Erweiterung”
  - keine “Eingriffe” in RACER erforderlich



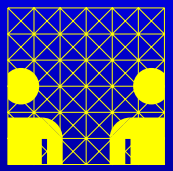
# Ausblick

- Haben:
  - Einfache, aber interessante RACER-”Erweiterung”
  - keine “Eingriffe” in RACER erforderlich
  - Interesse an der Erweiterung auch von anderen RACER-Nutzern bekundet



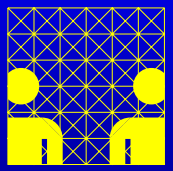
# Ausblick

- Haben:
  - Einfache, aber interessante RACER-”Erweiterung”
  - keine “Eingriffe” in RACER erforderlich
  - Interesse an der Erweiterung auch von anderen RACER-Nutzern bekundet
  - Nicht nur für RACER nutzbar (s. “GIS-Demo” im 2. Teil des Vortrages)



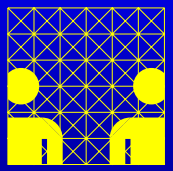
# Ausblick

- Zu tun:



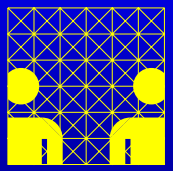
# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden



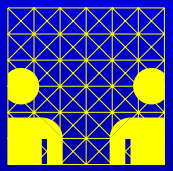
# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden
  - Vollständige Inferenz fürs Semantische Rewriting notwendig



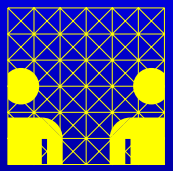
# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden
  - Vollständige Inferenz fürs Semantische Rewriting notwendig
  - Ausgeführt und optimiert werden können die Queries immer, auch ohne jegliche Inferenz-Maschine



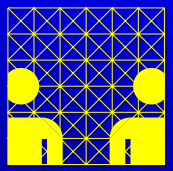
# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden
  - Vollständige Inferenz fürs Semantische Rewriting notwendig
  - Ausgeführt und optimiert werden können die Queries immer, auch ohne jegliche Inferenz-Maschine
  - “Query Repository” kommt prinzipiell auch mit unvollständiger Inferenz zurecht



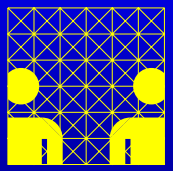
# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden
  - Vollständige Inferenz fürs Semantische Rewriting notwendig
  - Ausgeführt und optimiert werden können die Queries immer, auch ohne jegliche Inferenz-Maschine
  - “Query Repository” kommt prinzipiell auch mit unvollständiger Inferenz zurecht
  - Test auf sehr großen ABoxen



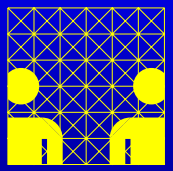
# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden
  - Vollständige Inferenz fürs Semantische Rewriting notwendig
  - Ausgeführt und optimiert werden können die Queries immer, auch ohne jegliche Inferenz-Maschine
  - “Query Repository” kommt prinzipiell auch mit unvollständiger Inferenz zurecht
  - Test auf sehr großen ABoxen
  - Fehlerbereinigung



# Ausblick

- Zu tun:
  - Inferenz mit NAF-Atomen noch nicht voll verstanden
  - Vollständige Inferenz fürs Semantische Rewriting notwendig
  - Ausgeführt und optimiert werden können die Queries immer, auch ohne jegliche Inferenz-Maschine
  - “Query Repository” kommt prinzipiell auch mit unvollständiger Inferenz zurecht
  - Test auf sehr großen ABoxen
  - Fehlerbereinigung
  - Nutzbarmachung für andere RACER-User



Danke
für Ihre
Aufmerksamkeit!